**ABSTRACT**

Our project entitled **“Mass Vigilant System”** reduces the conflicts, suspicious activities, malpractices when compared to the manual process. This project mainly takes less time to inform the respective authorities automatically with an alert.

In present situation, the information related to any malpractices or suspicious activities is given manually by the surrounding people to the authorities which requires more time and continuous monitoring.

In this proposed system, Firstly the Live video of detects the faces of people and their count. If the count of the faces is more than the specified count then it automatically raises an alarm alert to the respective authority. So, the authority can immediately react to the situation. Here Admin has an authority to add the users. If the admin adds the user then he can login and can monitor video and can able to listen alarm if any suspicious activity caught. But the user doesn’t have authority to add other user. Object identification is the first and essential step for object recognition and it is used to detect multiple objects(faces) in the images. The techniques used for object (FACE) detection are OpenCV Library and haarcascade frontal face Algorithm. MYSQL is used for Database and Python flask as server side technology. Visual Studio 10 is used as IDE.

**CHAPTER 1**

**INTRODUCTION**

**1 INTRODUCTION**

Mass Vigilant System is a web based application based on Python OpenCv Library. It detects the faces of the person. It counts the number of faces recognized in the live video and returns the count of the people basing on the faces and If the count of person is more than the specified count by the authority then the alarm will raises. When the alarm raises, the users or admin can see the live video and if any action is required then necessary steps is taken manually.

**1.1 Existing System :**

Existing System is a Manual process where we need continuous monitoring of video and which is very difficult.

**1.2 Proposed System :**

In our Proposed System the camera will detects and counts the faces in the live video. If the Head count is more then the Authority specified count then it will raises an alert to the respected person by Sound Alarm.

**Applications :**

* Hospitals
* Poling booths
* ATM’s

**CHAPTER 2**

**REQUIREMENTS ANALYSIS**

**2.1 System Requirements Specification**

This project is a web-based application. The hardware and software specifications of the system are mentioned below.

**2.1.1 Hardware Requirements**

Processor : Intel i3 or above Processor

RAM : 4GB

HDD : 1TB

Camera : HD 720P

**2.1.2 Software Requirements**

Front end(IDE) **:** MS Visual Studio 2010

Server Side Technology **:** Python Flask

Language **:** Python Open CV, HTML5,Java Script Backend(database) **:** MySQL

**2.1.3 Working Platform**

Operating Systems **:** Windows 7 or above Versions

**2.2 Software Requirement Specification**

## 2.2.1 Product Perspective

The Mass vigilant System is intended to replace the manual model of monitoring the camera and identifying the issue. The features expressed in this Software Requirements Specification document are intended to be fully implemented in version 1.0. The system will be developed in such a way to provide easy addition of enhanced features, which may be desired in subsequent versions.

## 2.2.2Product Functions

The main feature of this Mass Vigilant System is to take the count of faces in the crowd prohibited areas and sending alerts automatically to the respected authorities. This system detects number of faces in the image if the head count is more than the specified count it will send an alert message to the respected authority.

## 2.2.3 Operating Environment

OE-1: The Mass Vigilant System shall function on the PC provided by the management. This entails the system to operate on the Windows platform.

OE-2: The Mass Vigilant System shall interface between Windows designated to store the records of Admin and Users in the organization.

## 2.2.4 Design and Implementation Constraints

The programming is done using PYTHON,Python Flask for connection purpose.

## User Documentation

UD-1: The system will provide a user manual in a word document that describes the functionality and options available to the user.

## 2.2.6 Assumptions and Dependencies

## AS-1: The database mentioned within this Software Requirements Specification document is previously administered with the correct information needed by the Mass Vigilant System.

DE1: Statistics onMassVigilantSystem is dependenon organization’s consistent utilization of the system.

# 2.3 External Interface Requirements

## 2.3.1 User Interfaces

## All modifications to the database will be done through a keyboard.The interface would be viewed best using 1024 x768 and 800 x 600 pixels resolution setting.

## 2.3.2 Hardware Interfaces

## 4 GB of RAM with i3 or above version Processors and 780p camera are better to use for quality results.

## 2.3.3 Software Interfaces

Anaconda prompt to run the application. The Mass Vigilant System code will use Python language for coding. Python Flask for the database connection purpose.

## 2.3.4 Communications Interfaces

Mass vigilant System will communicate to the database through SQLite3.

# 2.4 System Features

The following are some of the features of the system.

## 2.4.1 Recognition

In this System the camera will detect the faces of the people in live, After detecting the faces it will check the head count in the picture.

## 2.4.2 Raising an Alert

Raising an alert is also one of the classes in this system. The count of the persons will be taken . if the Head count is more then the Admin specified count it will raise an alert to the respected persons by Sound Alarm.

**CHAPTER 3**

**SYSTEM DESIGN**

SYSTEM DESIGN phase follows analysis phase. Design is maintaining record of proof design divisions and providing a blueprint for the implementation phase. Design is the bridge between system analysis and system implementation.s

System design is transition from a user oriented, document oriented to programmers. The design is a solution, a “how to” approach to the creation of a new system. This is composed of several steps. It provides the understanding and procedural details necessary for implementing the system recommended in the feasibility study. Design goes through logical and physical stages of development; a logical design review the present physical system, prepare input and an output specification, detail the implementation plan, and prepares a logical design walkthrough.

**3.1 Design Methodology**

The design process for software system has two levels:

|  |
| --- |
| 1. System design or Top level design |
| 1. Detailed design or Logical design. |
|  |

**3.1.1 System Design**

In the system design the focus is on deciding which modules are needed for the system, the specification of these modules and how these modules should be interconnected.

**3.1.2 Detailed Design**

In detailed design the interconnection of the modules or how the specification of the modules can be satisfied is decided. Some properties for a software system design are

|  |
| --- |
| 1.Verifiability |
| 2.Completeness |
| 3.Consistency |
| 4.Traceability |
| 5.Simplicity / Understandability |

**3.2 DATA DICTIONARY**

After carefully understanding the requirements of the client the entire data storage requirements are divided into tables. The below tables are normalized to avoid any anomalies during the course of data entry.

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Description** | **Constraints** |
| Name | Varchar(50) | Name of the Admin | Unique |
| Username | Varchar(30) | User Id of the Admin | Primary |
| Password | Varchar(30) | Password of Admin | Not Null |

Database Table for Admin\_Login

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Description** | **Constraints** |
| Name | Varchar(30) | Name of the User | Unique |
| Email | Varchar(50) | Email of the User | Not Null |
| Contact\_no | Varchar(10) | Phone Number | Not Null |
| Username | Varchar(30) | User Id of the User | Primary |
| Password | Varchar(30) | Password of User | Not Null |

Database Table for User\_Login

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Description** | **Constraints** |
| Name | Varchar(50) | Name of the User | Not Null |
| Email | Varchar(60) | Email of the User | Not Null |
| Subject | Varchar(100) | About his Explanation | Primary |
| Comment | Varchar(1000) | Explanation regarding his/her findings. | Not Null |
| Database Table for Authenticated\_ Feedback | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Description** | **Constraints** |
| Name | Varchar(50) | Name of the User | Not Null |
| Email | Varchar(60) | Email of the User | Primary |
| Subject | Varchar(100) | About his Explanation | Not Null |
| Comment | Varchar(1000) | Explanation regarding his/her findings. | Not Null |
| Database Table for Feedback | | | |

**3.3 UML DIAGRAMS**

Unified Modeling Language is the language used to visualize, specify, construct and document any component of software engineering.

The Unified Modeling Language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic semantic and pragmatic rules.

A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram, which is as follows:

**User Model View**

|  |
| --- |
| 1. This view represents the system from the user’s perspective. 2. The analysis representation describes a usage scenario from the end users   Prespective. |

**Structural model view**

|  |
| --- |
| 1. In this model the data and functionality are arrived from inside the system. |
| 1. This model view models the static structures. |

**Behavioral Model View**

It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

**Implementation Model View**

In this the structural and behavioral as parts of the system are represented as they are to be built.

**Environmental Model View**

In this structural and behavioral aspects of the environment in which the system is to be implemented are represented. UML is specifically constructed through two different domains they are:

|  |
| --- |
| 1. UML analysis modeling, which focuses on the user model and structural model |
| views of the system. |
| 1. UML design modeling, which focuses on the behavioral modeling, implementation modeling and environmental model views. |
|  |

Every complex system is best approached through a small set of nearly independent views of a model; no single viewer is sufficient. Every model may be expressed at different levels of fidelity. The best models are connected to reality.

The UML includes nine such diagrams:

**Static Diagrams includes**

|  |
| --- |
| * Class diagram |
| * Object diagram |
| * Component diagram |
| * Deployment diagram |

**Dynamic diagrams includes**

|  |
| --- |
| * Use case diagram |
| * Sequence diagram |
| * Collaboration diagram |
| * State chart diagram |
| * Activity diagram |

**3.3.1 Use Case Diagram**

**U**se case diagrams are created to visualize the relationships between actors and use cases. A use case is a pattern of behavior the system exhibits. Each use case is a sequence of related transactions performed by an actor and the system. A use case diagram is a collection of various use cases and different actors. Use case

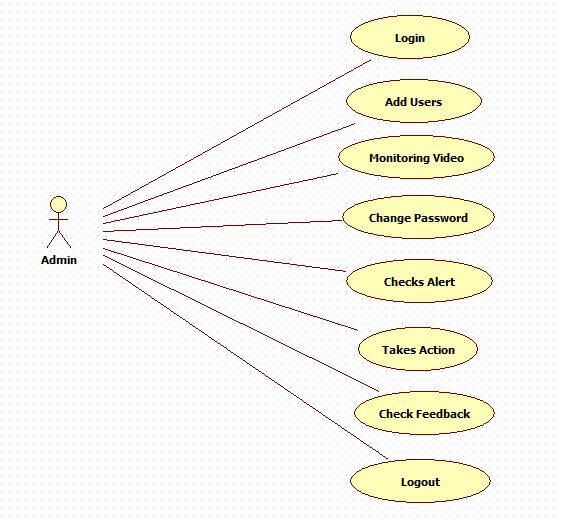
represents the verb form of a system .Use case is pictorially represented as elliptical form where the action performed by it is written inside the use case. For some systems use case has different types of relationships like includes and extends.

**Actor**

Actor in ause case diagram is **any entity that performs a role** in one given system. This could be a person, organization or an external system and usually drawn like skeleton.

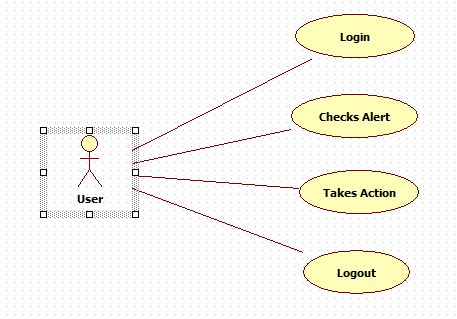
**Use Case**

A use case **represents a function or an action within the system**. It is drawn as an oval and named with the function.



Use case diagram for Admin

The above use case diagram specifies the procedure followed by Admin. The Admin has to login to the account by using his username and password. After login the admin dashboard will appears. There admin can add the users, change his/her password, monitors the video and also can check the feedback data. He/She also takes the action if the head count is high. Finally,the Admin will logout from the account.



Use case diagram for user

The above use case diagram specifies the procedure followed by User. User will login to the account by using the username and password given by the admin. User monitors the video and takes action if requires.

**3.3.2 Class Diagram**

A Class diagrams describe the static structure of a system, or how it is structured rather than how it behaves. Class diagram gives an overview of a system by showing its classes and the relationships among them. UML class is a rectangle divided into: class name, attributes, and operations. Our class diagram has three kinds of relationships.

* ***Association*** -- A relationship between instances of the two classes. There is an association between two classes if an instance of one class must know about the other in order to perform its work.
* ***Aggregation*** -- an association in which one class belongs to a collection. An aggregation has a diamond end pointing to the part containing the whole. In our diagram, Order has a collection of Order Details.
* ***Generalization*** -- an inheritance link indicating one class is a super class of the other.

Class diagrams are widely used to describe the types of objects in a system and their relationships.  Class diagrams model class structure and contents using design elements such as classes, packages and objects.  Class diagrams describe three different perspectives when designing a system, conceptual, specification, and implementation. These perspectives become evident as the diagram is created and help solidify the design.  This example is only meant as an introduction to the UML and class diagrams.  If you would like to learn more see the [Resources](http://atlas.kennesaw.edu/%257Edbraun/csis4650/A&D/UML_tutorial/resources.htm) page for more detailed resources on UML.

Classes are composed of three things: a name, attributes, and operations. Below is an example of a class.

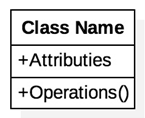
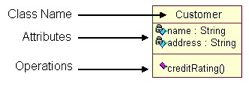
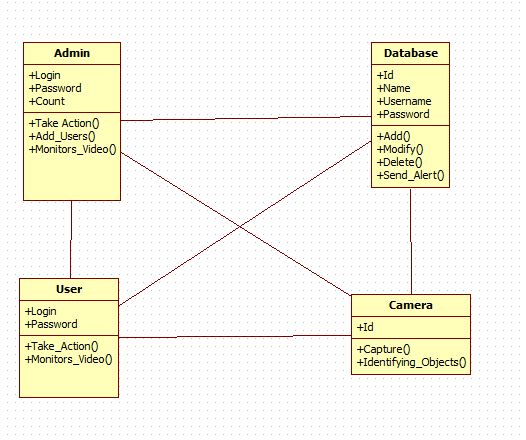


Fig 3.3.2.1Class Icon

Class diagrams are the mainstay of object-oriented analysis and design. UML class diagrams show the classes of the system, their interrelationships (including inheritance, aggregation, and association), and the operations and attributes of the classes. Class diagrams are used for a wide variety of purposes, including both conceptual/domain modeling

**

Sample Class Diagram

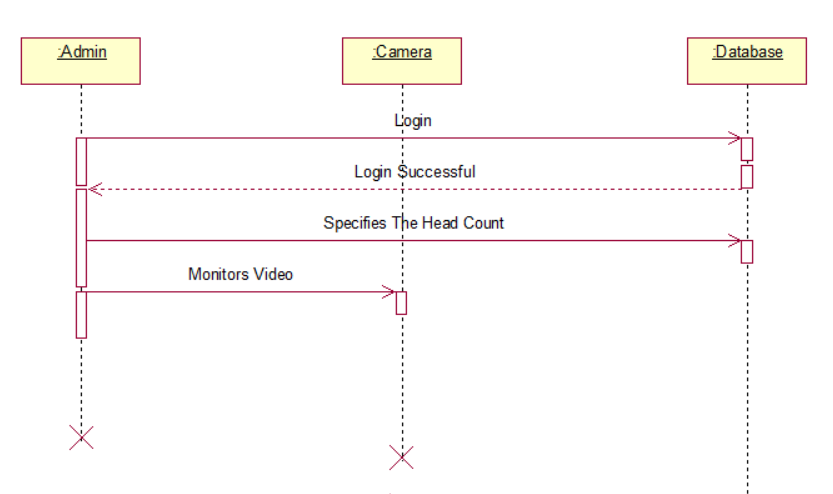


Class Diagram

A Class Diagram shows a set of classes, interfaces, collaborations and their Relationships. The class diagram contains 4 classes named Admin, Database, User, Camera. The admin and user login to their accounts. They monitors the video, if the alert came they takes the action based on the situation.

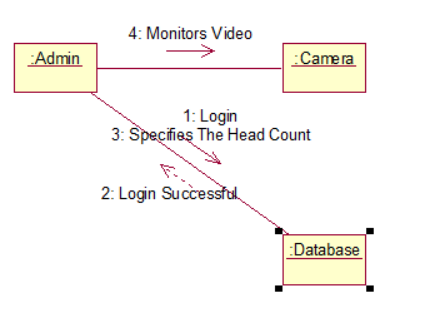
**3.3.3 Interaction Diagram**

A type of interaction diagram, a sequence diagram shows the actors of the object participating in an interaction and the events they generate arranged in a time sequence. Often a sequence diagram shows the events that results from a particular instance of a particular instance of a use case but a sequence diagram can also exist in a more generic form. The vertical dimension in a sequence diagram represents time, with time proceeding down the page the horizontal dimension represents different actors.



Sequence Diagram

In Sequence Diagram,The Admin will login to the website.He gives the head count.The dataset will identify the objects and takes the headcount.If the head count is more than the specified count it will send an alert message to the admin.The admin will respond according to it.



Collaboration Diagram

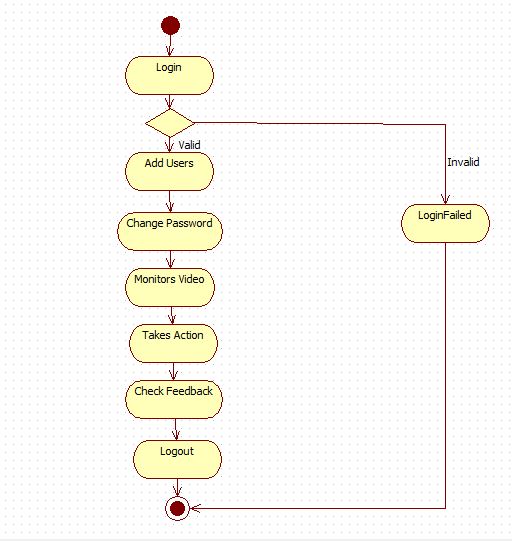
**3.3.4 Activity Diagram**

Activity diagram is another important diagram in UML to describe dynamic aspects of the system .Activity diagram is basically a flow chart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another operation.

The basic purposes of activity diagrams are similar to other four diagrams. It captures the dynamic behavior of the system. Other four diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another.

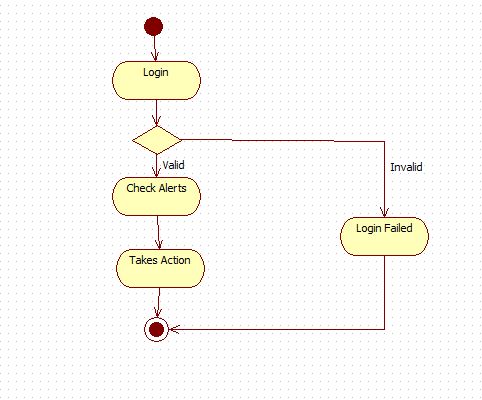
Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part.

It does not show any message flow from one activity to another. Activity diagram is some time considered as the flow chart. Although the diagrams looks like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.



Activity Diagram

The above Activity diagram specifies the procedure followed by Admin. In this Diagram, The Admin will login to the website. If he gives invalid credentials the login will failed and the process is stopped. If the login is successful then the admin dashboard will appear. There the Admin can add users, see feedback report, change password. If he gets an alert then admin will respond and takes action.



The above Activity diagram specifies the procedure followed by User. In this Diagram, The User will login to the website. If he gives invalid credentials the login will failed and the process is stopped. If the login is successful then the User dashboard will appear. He monitors the video and if he gets an alert then user will respond and takes action.

**CHAPTER 4**

**MODULES**

**4.1 Module Description**

Mass Vigilant System includes 4 basic modules. They are

1. Admin
2. User
3. Object Identification

**4.1.1 Admin**

**4.1.1.1 Login**

Admin can login into his account by giving his username and password. If the both the user name and password are valid then the admin would be able to access the account.

**4.1.1.2 User Registration**

Admin can add the users by giving the username and password which are choosen by the user. Admin can view and check details of the registered users.

**4.1.1.3 Vigilance**

Admin Will be able to monitors the video of the camera and based on the total count he takes the necessary action.

**4.1.1.4 Feedback Report**

Admin can be able to see the feedback report which are given by the users and the Admin can solve the issues.

**4.1.1.5 Password Change**

Admin can change the password of his account for the better security purposes.

**4.1.2 User**

**4.1.2.1 Login**

User can login into his account by giving the username and password which are provided by the admin. If the both the user name and password are valid then the user would be able to access the account.

**4.1.2.2 Vigilance**

User Will be able to monitors the video of the camera and based on the total count he takes the necessary action.

**4.1.2.3 Feedback**

User can be able to give the feedback. User can raise the issues and can give the feedback to the admin.

**4.1.3 Object Identification**

The Admin/User after assessing the camera it will identifies the objects and recognizes the faces of the people present on the footage.

**CHAPTER 5**

**SYSTEM IMPLEMENTATION**

**Run.py :**

from flask import Flask, render\_template, url\_for, request, session, redirect

import cv

import cv2

import os

import logging as log

import datetime as dt

import numpy as np

from PIL import Image

from time import sleep

import sqlite3 as sql

import mysql.connector

from flask\_mysqldb import MySQL

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

p=[0]

q=[0]

data=0

user\_id = ""

mydb = mysql.connector.connect(

host="localhost",

user="root",

passwd="",

database="crowd")

mycursor = mydb.cursor()

@app.route("/")

def home():

return render\_template("home.html")

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

def homes():

name = request.form['name']

email = request.form['email']

subject = request.form['subject']

comment = request.form['comment']

query = ("INSERT INTO feedback SET name=%s , email=%s, subject=%s, comment=%s ")

values = (name,email,subject,comment)

mycursor.execute(query,values)

mydb.commit()

return render\_template("layout.html", data1="Successfully Sent, Thanks for showing your Interest.")

@app.route("/login/")

def login1():

return render\_template("login.html")

@app.route('/adminlogin/')

def adminlogin():

return render\_template("adminlogin.html", data1="")

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

def adminlogin1():

username = request.form['username']

password = request.form['password']

query = ("SELECT \* FROM adminlogin WHERE username=%s and password=%s")

values = (username,password)

mycursor.execute(query,values)

data = mycursor.fetchall()

count = mycursor.rowcount

if count==1:

id = data[0][0]

global user\_id

user\_id = id

p[0]=1

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

else:

return render\_template("adminlogin.html", data1="Please Enter Valid Credentials")

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

def login():

username = request.form['username']

password = request.form['password']

query = ("SELECT \* FROM login WHERE username=%s and password=%s")

values = (username,password)

mycursor.execute(query,values)

data = mycursor.fetchall()

count = mycursor.rowcount

if count==1:

id = data[0][0]

global user\_id

user\_id = id

q[0]=1

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

else:

return render\_template("login.html", data1="Please enter valid details")

@app.route("/vigilance")

def capture():

if user\_id:

return render\_template("capture.html")

else:

return redirect(url\_for('login1'))

@app.route("/vigilances")

def captures():

cascPath = "haarcascade\_frontalface\_default.xml"

faceCascade = cv2.CascadeClassifier(cascPath)

log.basicConfig(filename='webcamdata.txt',level=log.INFO)

video\_capture = cv2.VideoCapture(0)

count = 0

while True:

if not video\_capture.isOpened():

print('Unable to load camera.')

sleep(5)

pass

# Capture frame-by-frame

ret, frame = video\_capture.read()

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

faces = faceCascade.detectMultiScale(

gray,

scaleFactor=1.1,

minNeighbors=5,

minSize=(30, 30) )

# Draw a rectangle around the faces

for (x, y, w, h) in faces:

cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

if count != len(faces):

count = len(faces)

log.info("faces: "+str(len(faces))+" at "+str(dt.datetime.now()))

if len(faces)>=0:

global data

data=str(len(faces))

print(data)

# Display the resulting frame

cv2.imshow('Video', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break;

# Display the resulting frame

cv2.imshow('Video', frame)

# When everything is done, release the capture

video\_capture.release()

cv2.destroyAllWindows()

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

def register():

username = request.form['username']

password = request.form['password']

confirm = request.form['confirm']

if password==confirm:

query = ("INSERT INTO login SET username=%s , password=%s")

values = (username,password)

mycursor.execute(query,values)

mydb.commit()

return render\_template("registration.html", data1="Successfully added")

else:

return render\_template("registration.html", data1="Password Mismatch")

@app.route('/registration')

def registration():

return render\_template('registration.html')

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

def admindashboard():

if p[0]:

if request.method=="POST":

p[0]=0

return render\_template('adminlogin.html')

return render\_template("admindashboard.html", data=data)

else:

return render\_template('adminlogin.html')

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

def feedbacks():

query = ("SELECT \* FROM feedback")

mycursor.execute(query)

data = mycursor.fetchall()

return render\_template('feedback.html', data=data)

@app.route('/passwordchange')

def passwordchanges():

return render\_template('passwordchange.html', data=data)

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

def password():

password = request.form['password']

confirm = request.form['confirm']

if password==confirm:

sql = ("UPDATE adminlogin SET password = %s WHERE username = %s")

val = (password, "admin")

mycursor.execute(sql, val)

mydb.commit()

return render\_template("passwordchange.html", data1="Successfully Updated your Current Password !!")

else:

return render\_template("passwordchange.html", data1="Password Mismatched, Try Again !!!")

@app.route('/about')

def about():

return render\_template('about.html')

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

def camera():

return render\_template("camera.html", data=data)

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

def userdashboard():

if q[0]:

if request.method=="POST":

q[0]=0

return render\_template('login.html')

return render\_template("userdashboard.html", data=data)

else:

return render\_template('login.html')

@app.route('/thanq')

def thanq():

return render\_template('thanq.html')

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

app.run(debug=True)

**CHAPTER 6**

**TEST CASES**

**6.1 Software Testing**

In any software development, testing is a process to show the correctness of program and it needs the design specifications. Testing is needed to prove correctness completeness, to improve the quality of the software and to provide the maintenance aid. Some testing standards are therefore necessary to ensure completeness of testing, improve the quality of software and reduce the testing costs and to reduce study needs and operation time.

**6.2 Goals of Testing**

The following are goals of testing…

1. Testing is a process of executing a program with the intent of finding error.

2. A good test case is the one that has a high probability of finding an as at undiscovered error.

3. A successful test is one that uncovers an as at undiscovered error.

**6.3 Testing Methodology**

**6.3.1 Black box testing**

Black Box Testing is the testing process in which tester can perform testing on an application without having any internal structural knowledge of application. Usually Test Engineers are involved in the black box testing.

**6.3.2. White box testing**

White Box Testing is the testing process in which tester can perform testing on an application with having internal structural knowledge. Usually the developers are involved in the white box testing.

**6.3.3. Gray box testing**

Gray Box Testing is the process in which the combination of black box and white box techniques is use.

**6.4 Levels of Testing**

**6.4.1. Unit testing**

Individual components are tested to ensure that they operate correctly. Each component is tested independently without other system components.

**6.4.2. System testing**

The sub-systems are integrated to make up the entire system. The testing process is concerned with finding errors, which result from un-anticipated interactions between subsystem components.

**6.4.3. Integration testing**

Sometimes global data structures can represent the problems to uncover errors that are associated with interfacing the objective is to make unit test modules and built a program structure that has been detected by design.

**6.4.4. Acceptance testing**

This is the final stage in the testing process before the system is accepted for operational use. Acceptance testing may reveal errors and omissions in the system requirements definition because real data exercises the system in different ways from the test data.

**6.4.5. Regression testing**

Regression testing is actually that helps to ensure changes that don’t introduce unintended behavior as additional errors. Regression testing may be conducted manually by executing a subset of all test cases or using automated capture play back tool.

**6.5 Unit Test Cases**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TC**  **ID** | **Input** | **Description** | **Expected Results** | **Actual Result** | **Pass/Fail** |
| TC\_AL\_01 | Blank Username and Password | When Admin enters blank username and password. | Admin Dashboard Opened | Fill in the field. | Fail |
| TC\_AL\_02 | Blank Username and Filled  Password | When Admin enters blank username and filled password. | Admin Dashboard Opened | Fill in the field. | Fail |
| TC\_AL\_03 | Filled Username and Blank Password | When Admin enters  filled username and blank password. | Admin Dashboard Opened | Fill in the field. | Fail |
| TC\_AL\_04 | Incorrect Username and Password | When Admin enters  incorrect username  and password. | Admin Dashboard Opened | Enter Valid Credentials | Fail |
| TC\_AL\_05 | Correct Username and incorrect password | When Admin enters  correct username  and incorrect password. | Admin Dashboard Opened | Enter Valid Credentials | Fail |
| TC\_AL\_06 | Incorrect Username and correct password | When Admin enters  incorrect username  and correct password. | Admin Dashboard Opened | Enter Valid Credentials | Fail |
| TC\_AL\_07 | Username Filled Password Filled | When Admin enters  Correct username and password. | Admin Dashboard Opened | Admin Dashboard Opened | Pass |

**Admin Login**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TC**  **ID** | **Input** | **Description** | **Expected Results** | **Actual Result** | **Pass/Fail** |
| TC\_PC\_01 | password blank  confirm password blank | Blank password and confirm password is given by the admin. | Password changed | Fill in the field. | Fail |
| TC\_PC\_02 | password filled confirm password blank | Filled password and blank confirm password is given by the admin. | Password changed | Fill in the field. | Fail |
| TC\_PC\_03 | password blank  confirm password filled | blank password and filled confirm password is given by the admin. | Password changed | Fill in the field. | Fail |
| TC\_PC\_04 | Different password and confirm password | Different password and confirm password is given by the admin. | Password changed | Password mismatch | Fail |
| TC\_PC\_05 | Same password and confirm password | Same password and confirm password is given by the admin.. | Password changed | Password changed | Pass |

**Password change**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TC**  **ID** | **Input** | **Description** | **Expected Results** | **Actual Result** | **Pass/Fail** |
| TC\_UL\_01 | Blank Username and Password | When User enters blank username and password. | User Dashboard Opened | Fill in the field. | Fail |
| TC\_UL\_02 | Blank Username and Filled  Password | When User enters blank username and filled password. | User Dashboard Opened | Fill in the field. | Fail |
| TC\_UL\_03 | Filled Username and Blank Password | When User enters  filled username and blank password. | User Dashboard Opened | Fill in the field. | Fail |
| TC\_UL\_04 | Incorrect Username and Password | When User enters  incorrect username  and password. | User Dashboard Opened | Enter Valid Credentials | Fail |
| TC\_UL\_05 | Correct Username and incorrect password | When User enters  correct username  and incorrect password. | User Dashboard Opened | Enter Valid Credentials | Fail |
| TC\_UL\_06 | Incorrect Username and correct password | When User enters  incorrect username  and correct password. | User Dashboard Opened | Enter Valid Credentials | Fail |
| TC\_UL\_07 | Username Filled Password Filled | When User enters  Correct username and password. | User Dashboard Opened | User Dashboard Opened | Pass |

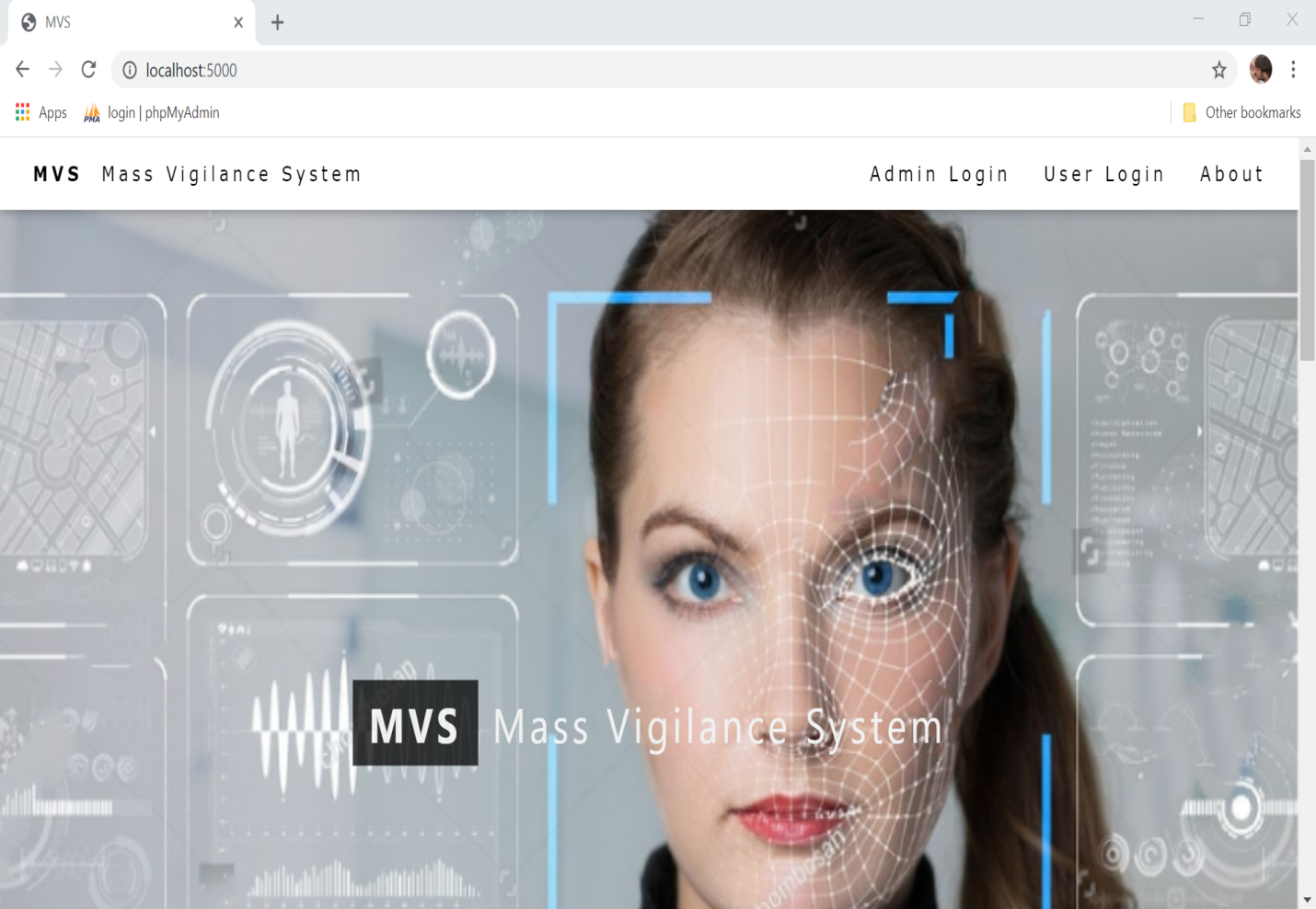
**User login**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TC**  **ID** | **Input** | **Description** | **Expected Results** | **Actual Result** | **Pass/Fail** |
| TC\_UR\_01 | Blank Username,  Password and confirm password | When User enters blank username and password. | User Successfully added. | Please fill in this field. | Fail |
| TC\_UR\_02 | Blank Username,  Password  and filled  confirm password | When User enters blank username and filled password. | User Successfully added. | Please fill in this field. | Fail |
| TC\_UR\_03 | Blank Username,  Confirm Password  and filled  password | When User enters  filled username and blank password. | User Successfully added. | Please fill in this field. | Fail |
| TC\_UR\_04 | Blank  Password,  confirm password and filled username. | When User enters  incorrect username  and password. | User Successfully added. | Please fill in this field. | Fail |
| TC\_UR\_05 | Filled Username and Different password and confirm password | When User enters  correct username  and incorrect password. | User Successfully added. | Password mismatch. | Fail |
| TC\_UR\_06 | Filled Username and same password and confirm password | When User enters  incorrect username  and correct password. | User Successfully added. | User Successfully added. | Pass |

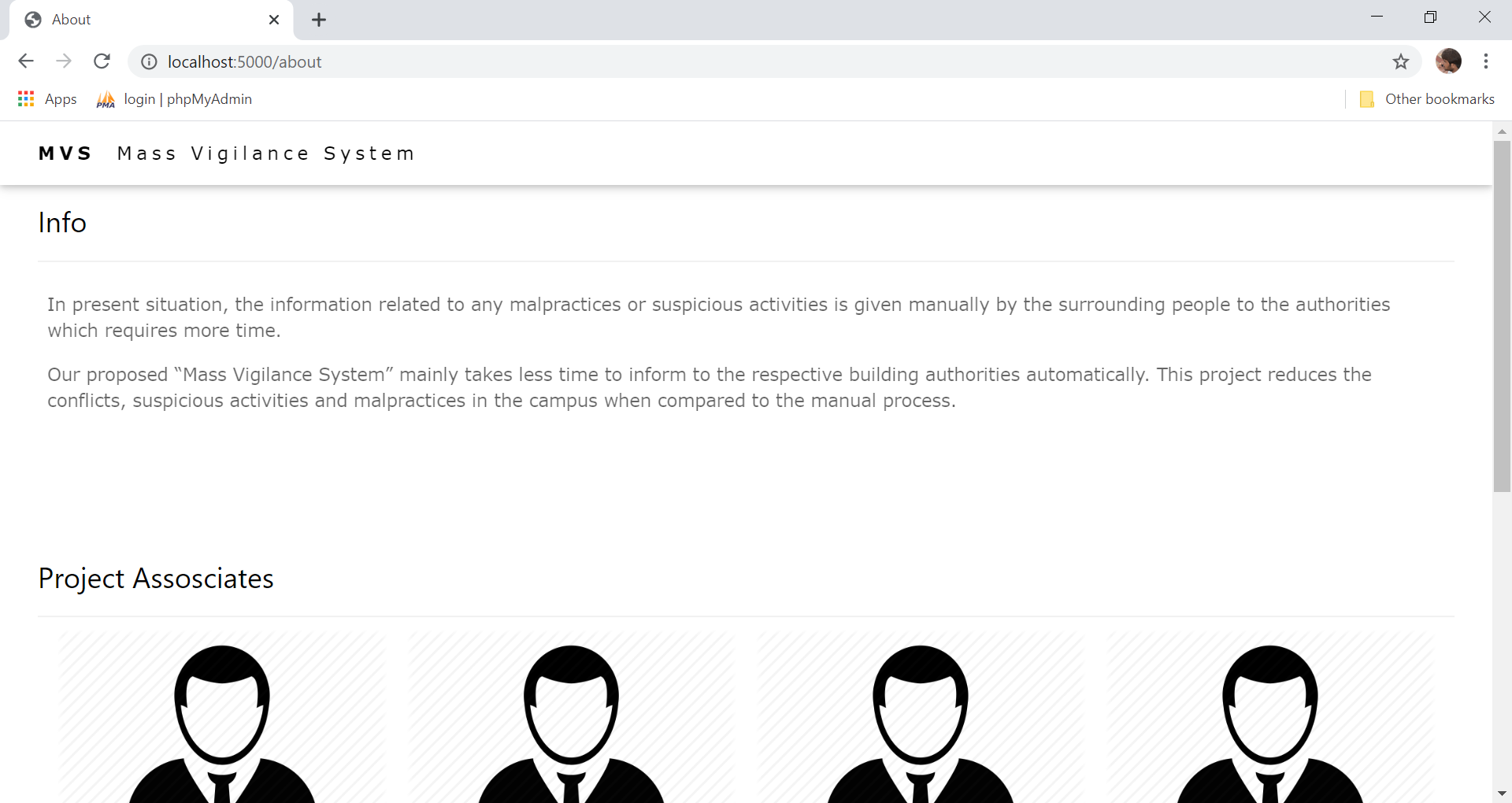
**User Registration**

**CHAPTER 7**

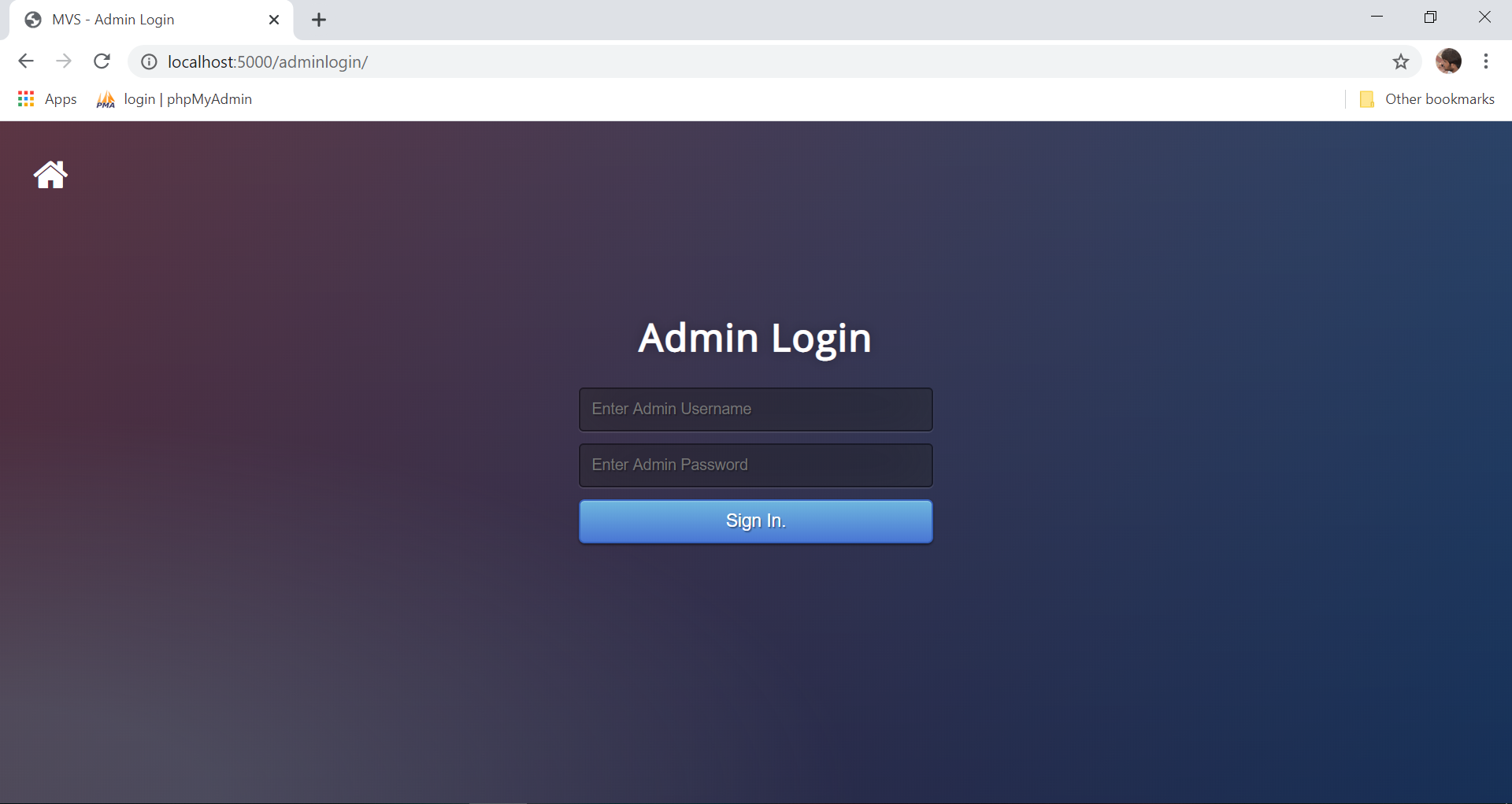
**SCREENSHOTS**



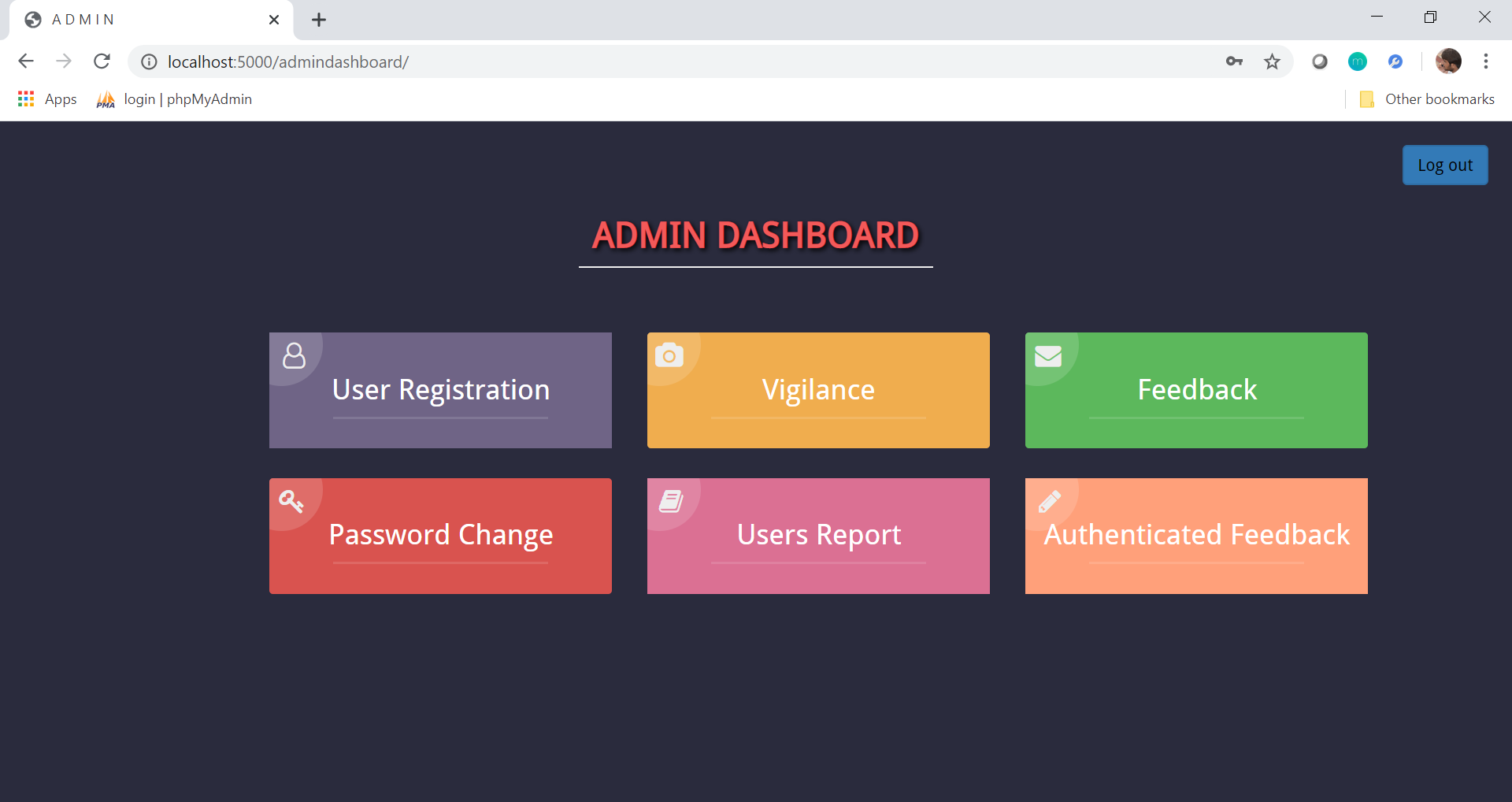
**Screen 7.1** Title page



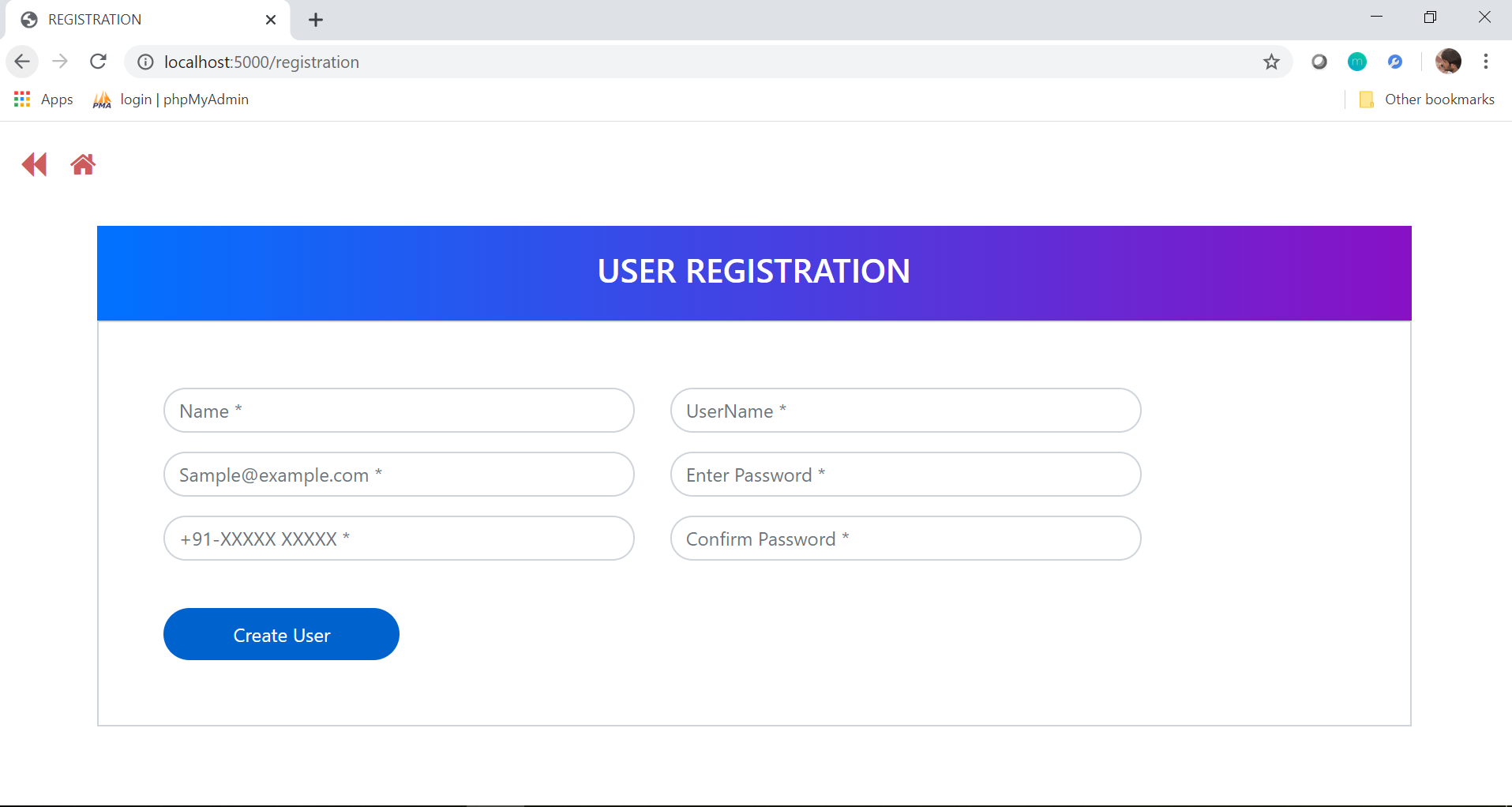
**Screen 7.2** About Project



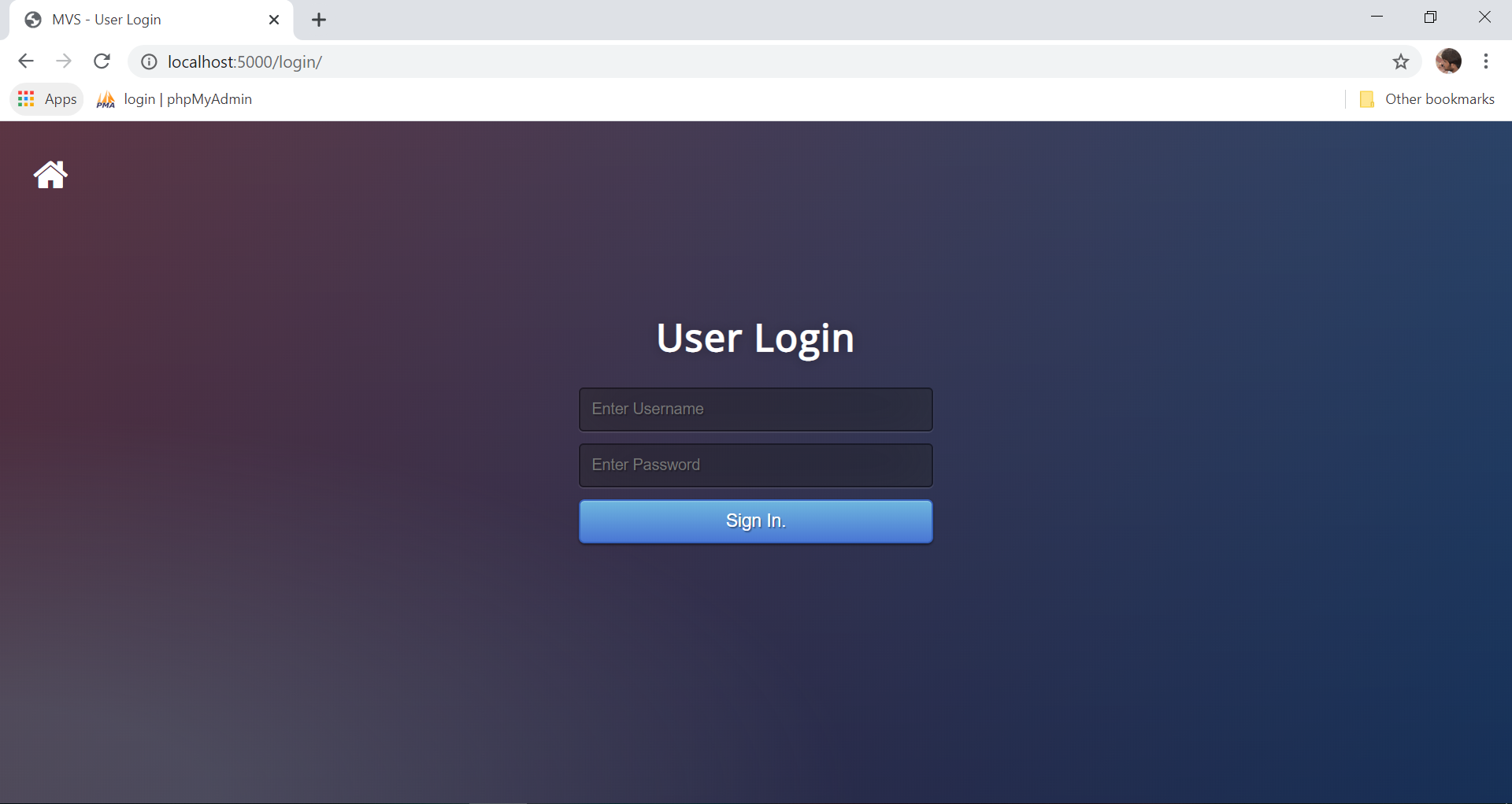
**Screen 7.3** Admin Login



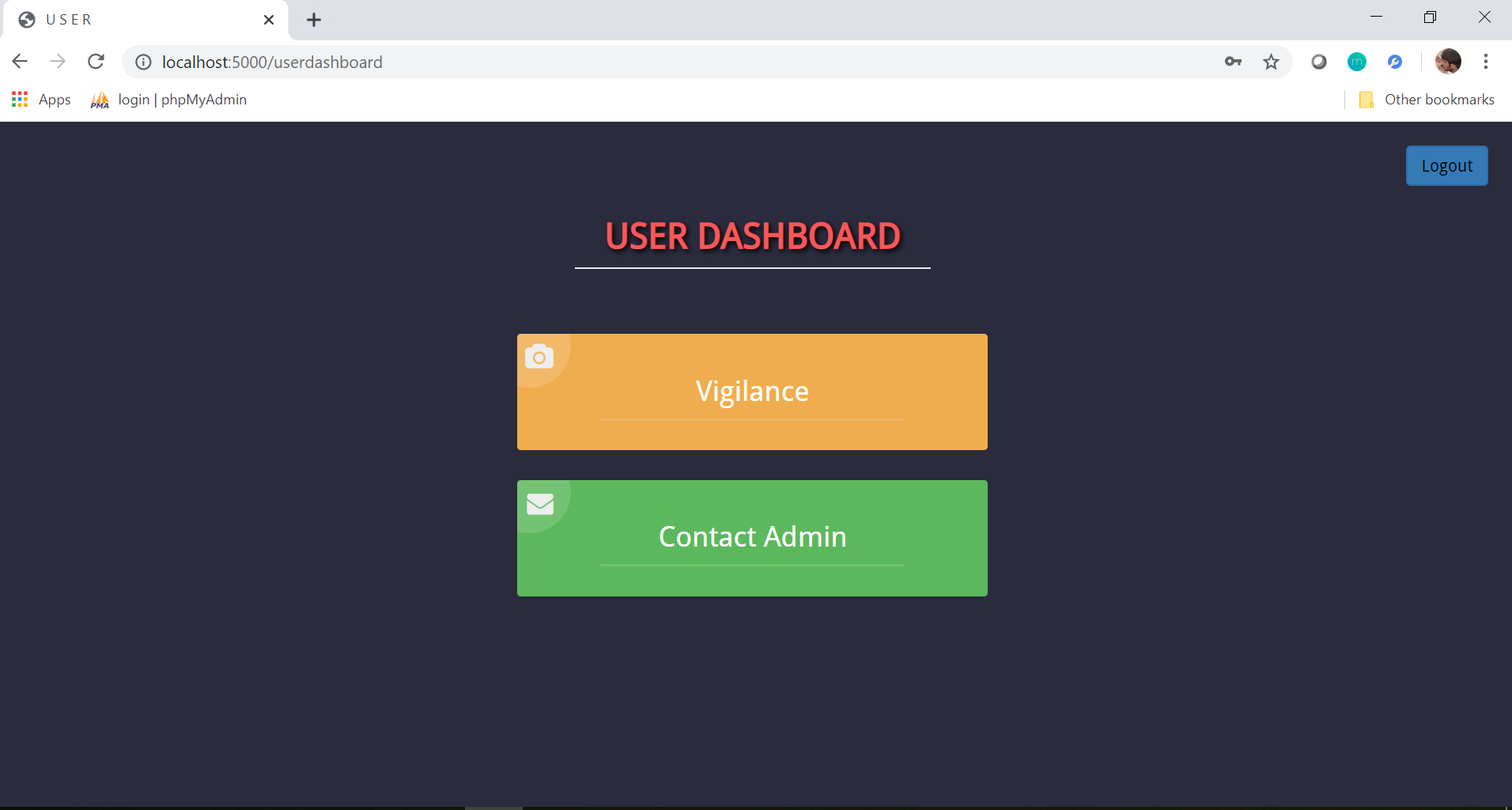
**Screen 7.4** Admin Dashboard



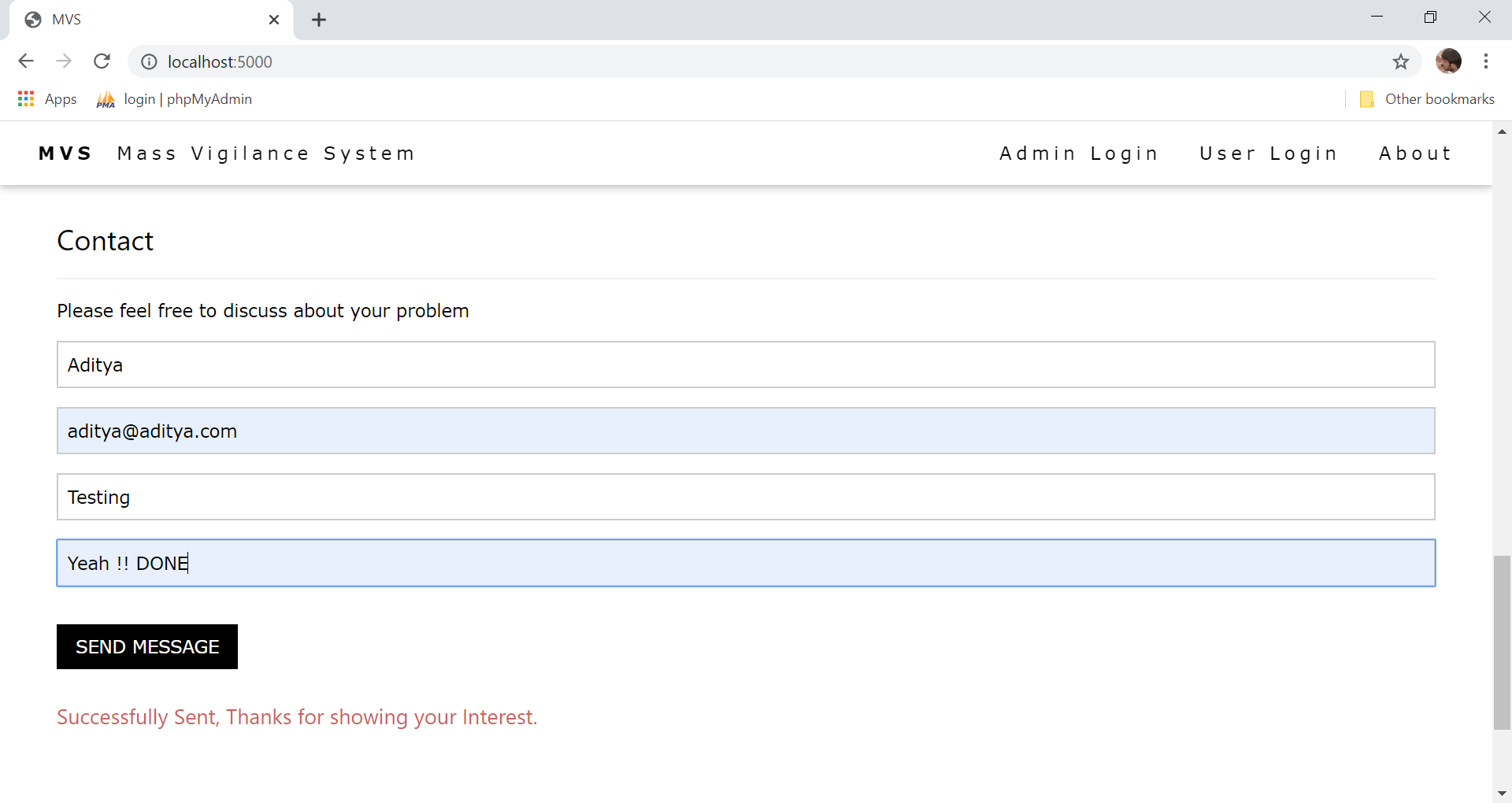
**Screen 7.5** User Registration



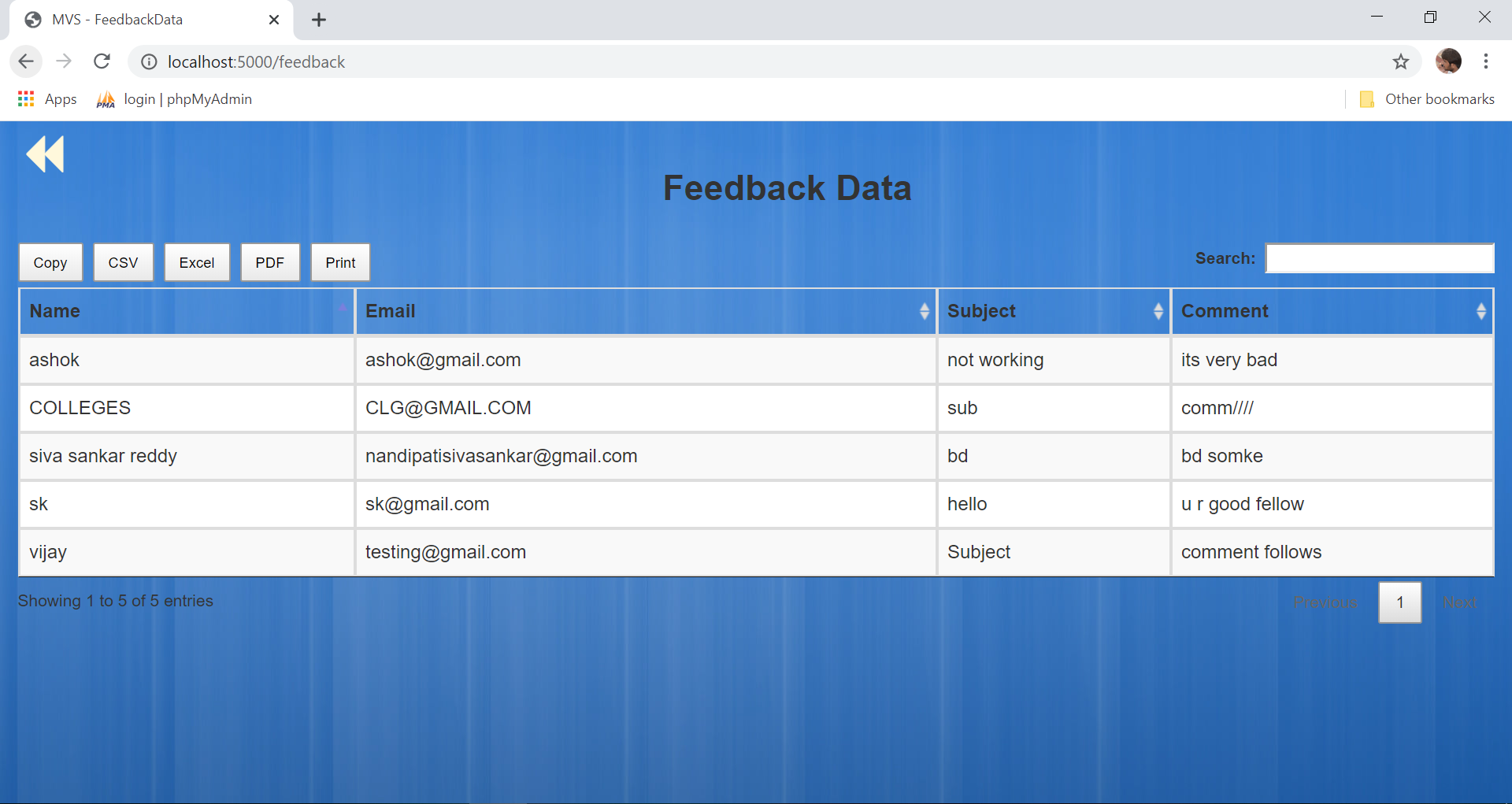
**Screen 7.6** User Login



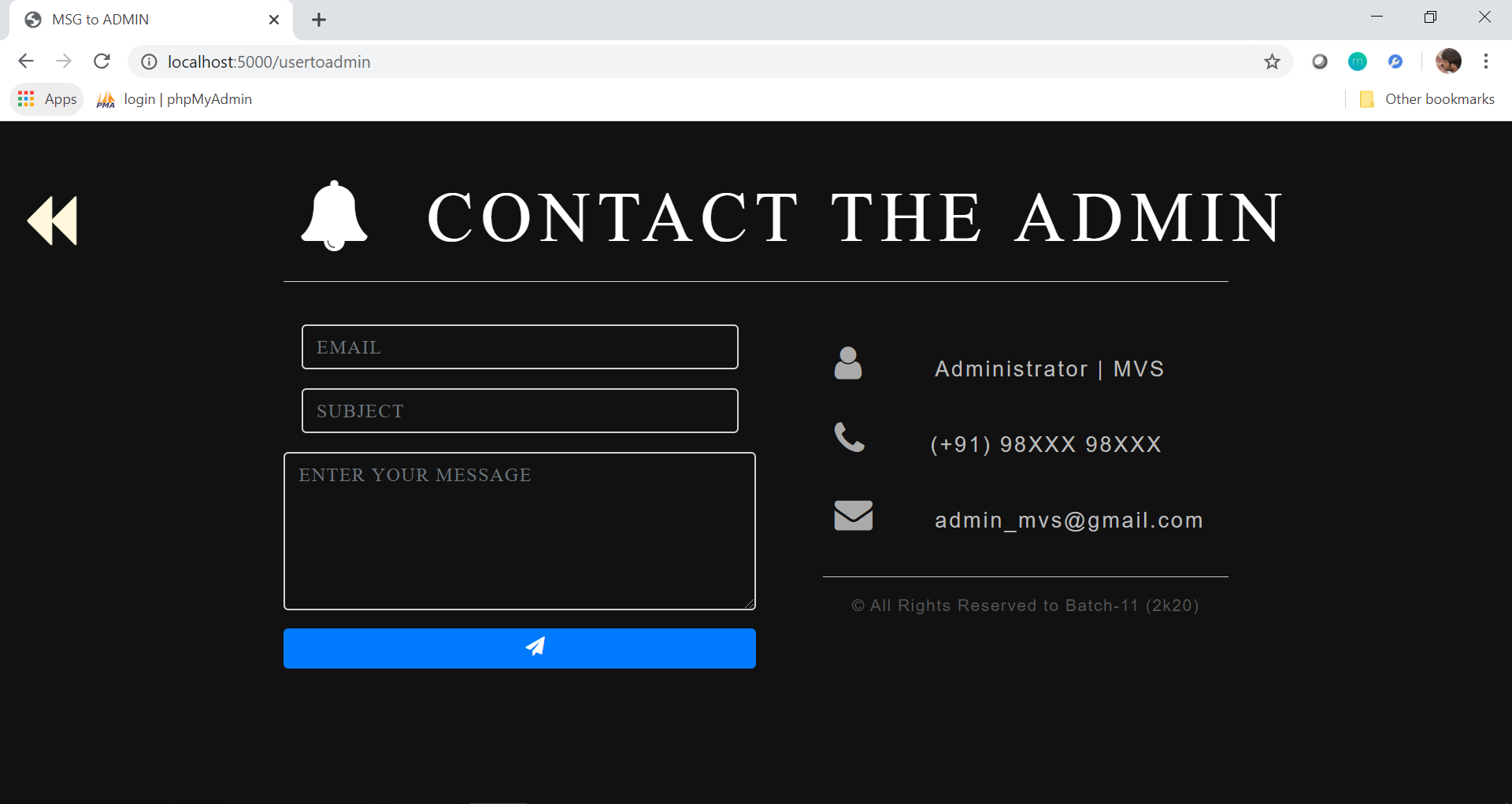
**Screen 7.7** User Dashboard



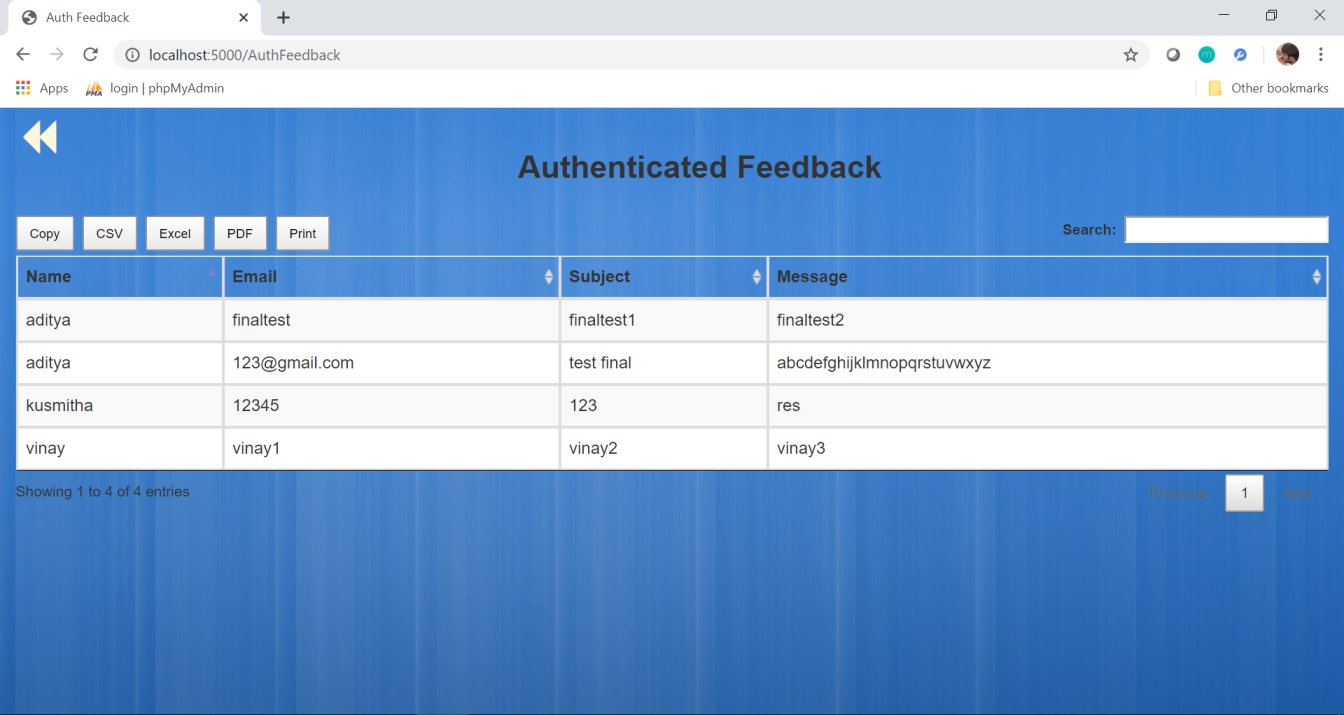
**Screen 7.8** Feedback Form



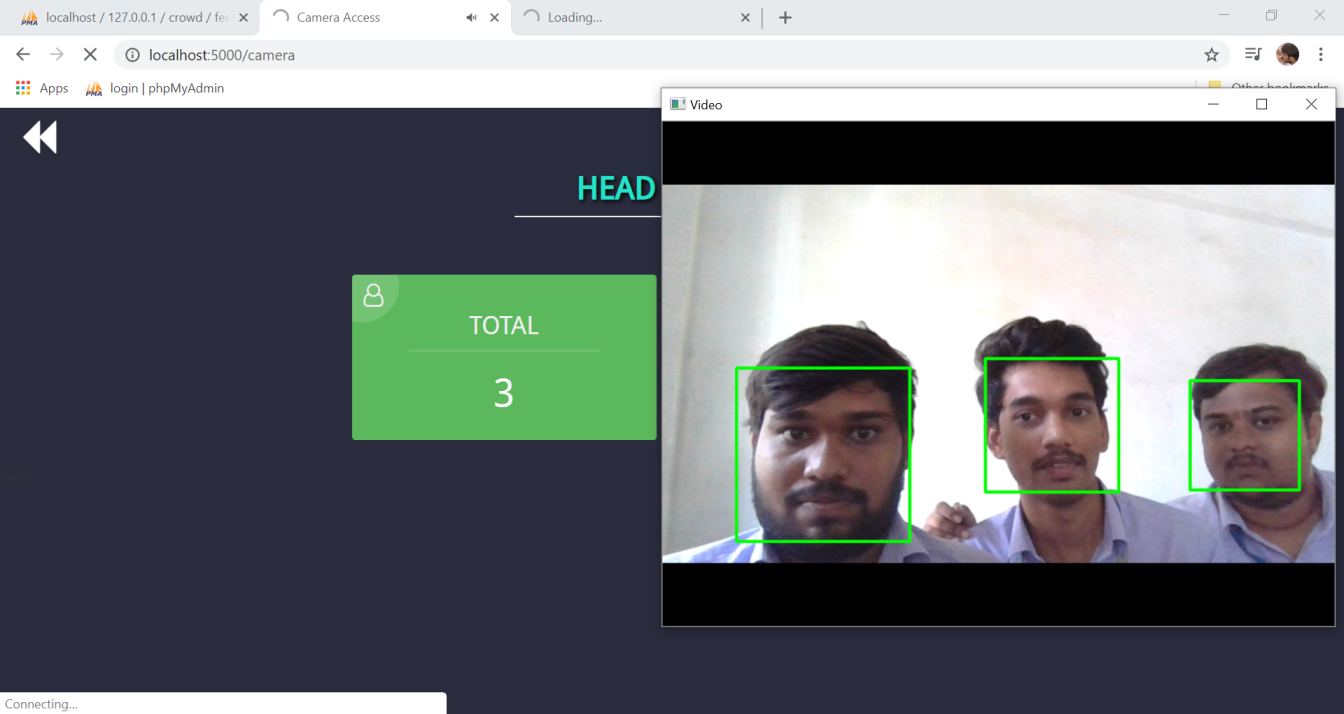
**Screen 7.9** Anonymous Feedback Data



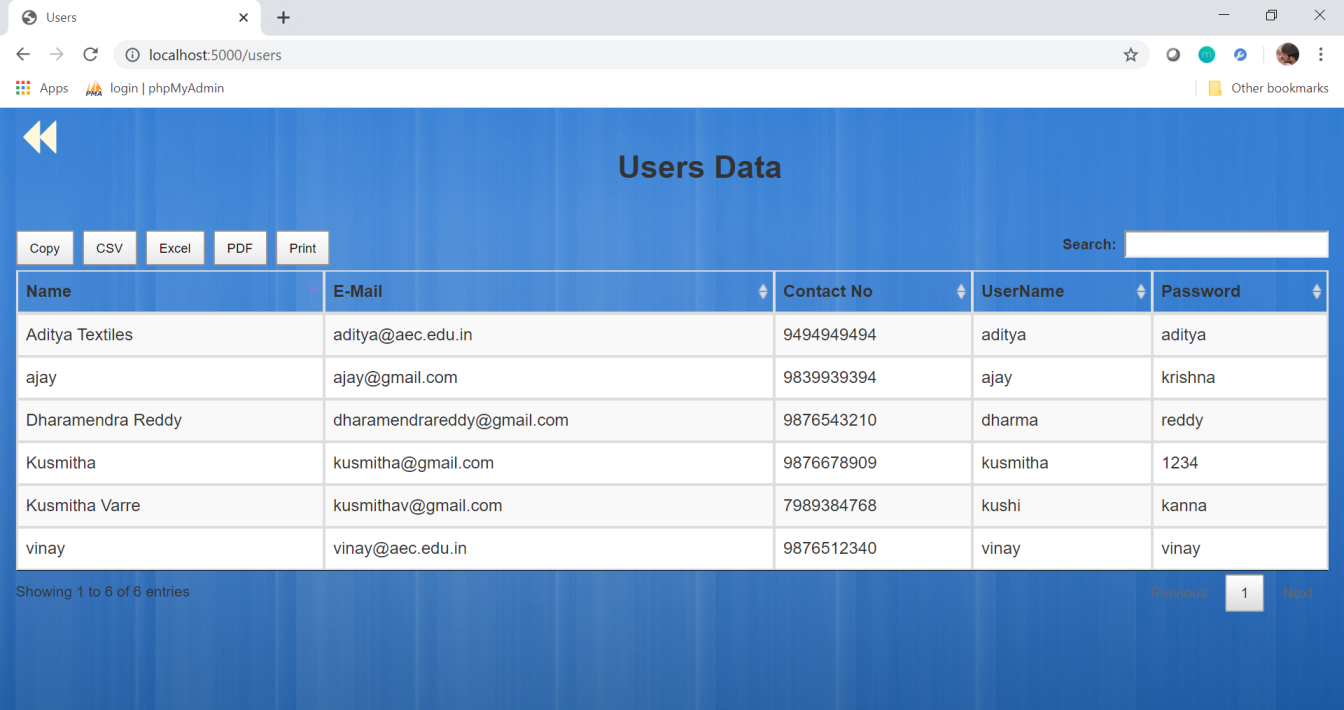
**Screen 7.10** Contact the Admin



**Screen 7.11** Authenticated Feedback



**Screen 7.12** Face Recogniton and Counting of Faces



**Screen 7.13** Users Data

**CHAPTER 8**

**CONCLUSION & FUTURE SCOPE**

**8.1 Conclusion**

In our project **“Mass Vigilant System”** is to find the suspicious activates are done by electronically rather then finding in manual. It reduces the manual work, man power and gives correct result. This project is being developed mainly for the polling booths, Hospital ICU’s, ATM’s,. Everyday actions are being handled electronically these days, instead of pen and paper. In this, we propose a system that takes the count of the faces identified in camera. It check the count of faces with the admin specified count, admin can specifies the count and Admin can change the count. If the face count is more then the admin specified count it will send an alert message to the respected person. Admin can add the users and admin can change his password. Users can give the feedback to the admin and admin can see the feedback.

* 1. **Future Scope**

In the next update version of **“Mass Vigilant System”** Admin can train the faces of people who are working on the organization. Now camera will count the number of faces and check whether the person is normal person or management of the organization. If the any of the face belongs to the management it will reduce the count based on that, still the count is more than the specified count it will send an alert message to Respected person.

**CHAPTER 9**

**BIBLIOGRAPHY**

**Sites Referred**

<https://w3schools.org/>

<https://stackoverflow.com/questions/>

<https://www.quora.com/HeadCountUsingOpenCV>

<https://docs.anaconda.com>

<https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/>