Prompt Engineering for Web pages

-Mihir Yashwant Ingole

HTML pages:

Corresponding CSS files also need to be generated.

1. index.html

index.html has the below code.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <link rel="stylesheet" href="static/css/styles.css">
  <title>Dummy Website</title>
</head>
<body>
  <header>
    <h1>Dummy Website</h1>
  </header>
  <nav>
    <ul>
      <li><a href="#home">Home</a></li>
      <li><a href="#about">About</a></li>
      <li><a href="#contact">Contact</a></li>
     <li><a href="{{ url_for('login') }}">Login</a></li>
    </ul>
  </nav>
  <main>
    <section id="home">
      <h2>Welcome to the Dummy Website</h2>
      <img src="static/images/home.jpg" alt="Home image" />
      <p>This is the home section. You can add any content you'd like
here.</p>
```

```
        </section>
        <section id="about">
          <h2>About Us</h2>
          <img src="static/images/about.jpg" alt="About image" />
          <p>Learn more about our dummy website and the services we
    provide.</p>
        </section>
        <section id="contact">
          <h2>Contact Us</h2>
          <img src="static/images/contact.jpg" alt="Contact image" />
          <p>Get in touch with us via email or our social media
    channels.</p>
        </section>
      </main>
      <footer>
        <p>&copy; 2023 Dummy Website. All rights reserved.</p>
      </footer>
    </body>
    </html>
```

2. login.html
   login.html has the below code and further code may need to be added to redirect to user page when the authentication of the user is successful.
   Code needs to be added to show if the user authentication has failed.
   Code needs to be added to the below code to add an option of “Continue as guest” which will redirect a user to another page called "guest.html”.
   guest.html has the description written below.
   The login has an option to “Register”. This should be redirected to the register.html page. The register.html page is described further.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <link rel="stylesheet" href="static/css/login.css">
  <title>Login</title>
</head>
<body>
  <div class="login-container">
    <h2>Login</h2>
    <form action="/login" method="post">
      <div class="form-group">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required>
      </div>
      <div class="form-group">
        <label for="password">Password:</label>
        <input type="password" id="password" name="password"
required>
      </div>
      <button type="submit">Login</button>
    </form>
    <p>Don't have an account? <a href="/register">Register</a></p>
  </div>
</body>
</html>
```

3. register.html
   The option in the login.html page called “Register” redirects to another page called register.html.
    a. “First name” field has a blank in front of which take characters of first name as input.
    b. “Last name” field has a blank in front of which take characters of last name as input.
    c. “Username” field has a blank in front of which take characters of username as input.
    d. “Password” field has a blank in front of which take characters of password as input.
    e. “Re-type Password” field has a blank in front of which take characters of password as input. The password entered in this field should be same as the password entered in the previous “Pass word” field.

4. user.html
   user.html file has 2 options.
    a. “Perform Steganography” button.
    b. “Gallery” button.

   Perform Steganography Option redirects to a page called steganography.html and
   Gallery option redirects to a page called gallery.html.
   Both steganography.html and gallery.html are described below.

5. guest.html
   guest.html has the following: -
   a. “First Name”: This takes the first name of user as input.
   b. “Last Name”: This takes the last name of user as input.

   If no inputs are provided for either option then it displays an error message and requests to try again.

6. steganography.html
   This html file has the below options.
   a. “Upload message file” button which takes a file of format JPG, MP4, OGG, MPG, AVI, MOV, WAV, DOC or text as input. File length cannot exceed 10 MB.
   b. “Upload Plaintext (Carrier) File” button which takes a file of format JPG, MP4, OGG, MPG, AVI, MOV, WAV, DOC or text as input. File length cannot exceed 10 MB.
   c. “Starting bit” option with a blank in front which takes an integer value or throws an error message.
   d. “Mode (fixed, variable)” option with a blank in front which takes only values “fixed” or “variable” or else throws an error message asking to enter only either of the two values.
   e. “Length of replacement” option with blank in front which takes an integer value or throws an error message. This field only appears when the input to the “Mode (fixed, variable)” field had an input of “fixed”.

7. gallery.html
   This displays all the content which is stored in one after another.

Below are the requirements for the assignment.

Description:

Create a web “site” (interface, service) that allows a user to submit a file, such as a picture, and a secret message

and you will “hide” that secret message inside the file using steganography. Then you will “post” that file (if a picture, you can display it) on a publicly accessible web site. Your web service should have user authentication

(if submitting for steganography) but anyone (no authentication) may look at postings.

Steganography Details:

The user will give a plaintext file which can be any format (P), a message (M) that may also be of any format,

and three additional parameters (S) the starting bit number, (L) the length (actually the periodicity) of the replacement

(in bits) and (C) the chosen mode of operation.

1. Given a message (M) which may be of any format (commonly a text, JPG, MPG, or similar) which will be

the message we wish to “hide”.

2. And, given a file (P) which will act as a carrier, one wishes to “embed” a message (M – the payload) by

“modulating” (changing) the contents of the carrier (P).

3. With the carrier (P) (this is the “plaintext” carrier) which is length LenP bits, one wants to change every Lth bit,

(where L is supplied by the user).

Every Lth bit is replaced by successive bits from M, the message.

4. Frequently, we wish to skip S bits at the beginning of P, because of the format or “type” of P

(otherwise P will appear “corrupted”).

5. A simple enhancement would allow L to change during processing (L = 8, then 16, then 28, then 8 again, etc.),

which will be specified by the mode (C).

6. Both the message (M) and the plaintext (P) may be of any format (commonly a JPG, MP4, OGG, MPG, AVI,

MOV, WAV, DOC, text or similar)

7. This process should be reversible, to be able to retrieve the original message.

Web Service (Site):

Any users may view (or access) the modified files, but only authenticated users may

submit the plaintext files, the “hidden” message, and associated information.

Based on all the above please modify the flask code below so that all the above requirements are satisfied for steganography, deployment on MS Azure cloud and storage in SQL database.

```
from flask import Flask, render_template, request

import pyodbc


app = Flask(__name__)


app=Flask(__name__,template_folder='templates') #CHANGE WHEN ADDING
FRONTPAGE


#CONNECTION STRING  SQL

#driver= '{ODBC Driver 13 for SQL Server}'

driver= '{ODBC Driver 17 for SQL Server}'

server = 'tcp:xxxxxxxxxxxxsqldb.database.windows.net,1433'

database = 'XXXXXXXXXXXXXXXX'

username = 'xxxxxxxxxxxxxxxx'

password = '************'

#conn = pyodbc.connect('DRIVER={SQL
Server};SERVER='+server+';DATABASE='+database+';UID='+username+';P
WD='+ password)

conn =
pyodbc.connect('DRIVER='+driver+';SERVER='+server+';DATABASE='+data
base+';UID='+username+';PWD='+ password+';')

cursor = conn.cursor() #cursor

cursor.execute('''select * from users''')
```

```
rows = cursor.fetchall()


@app.route('/')
def index():
    return render_template("index.html")


@app.route('/login')
def login():
   return render_template('login.html')


@app.route('/login/userlogin',methods=['POST'])
def userlogin():
        if request.method=='POST':
            details=request.form.get
            username=details("username")
            password=details("password")
            cursor.execute('''Select username from users where username = ?
and password =? ;''',username,password)
            name = cursor.fetchone()
        return render_template('homepage.html',name=name)


if __name__ == '__main__':
    app.run(debug=True)
```

8. encryption.html
   This html file has the below options.
      f. "Upload message file" button which takes a file of format JPG, MP4, OGG, MPG, AVI, MOV, WAV, DOC or text as input. File length cannot exceed 10 MB.

    g. “Encryption type (AES, RSA)” option with a blank in front which takes only values “AES” or “RSA” or else throws an error message asking to enter only either of the two values.
    h. “Block Mode” option with a blank in front which takes only values “CBC” or “OFB” or “CFB” else throws an error message asking to enter only either of these values. This field only appears when the input to the “Encryption type (AES, RSA)” field had an input of “AES”.
    i. “Receiver’s username” option with a blank in front which takes the username of the receiver which is a string of character.

9. encryption_output.html
    a. “Download receiver’s public key” button to download the receiver’s public key which is provided from the backend.
    b. “Download Encrypted File” button to download the encrypted file which is provided from the backend. This field only appears after entering the all the details till “Receiver’s username”.
    c. “Download Decrypted File” button to download the decrypted file which is provided from the backend. This field only appears after entering the all the details till “Receiver’s username”.
    d. “Hash Original” field will show the hash generated from the original file which will be provided from the backend.
    e. “Hash Decrypted” field will show the hash generated from the decrypted file which will be provided from the backend.
    f. If “Hash Original” and “Hash Decrypted” field has exactly the same value, then “Hash Comparison is Successful” is printed else “Hash Comparison Failed” is printed.