**QR-Code Generator Using Streamlit**

**Abstract**

This report describes the development of a QR-Code generator application using Python and the Streamlit library. The application is designed to generate QR codes for both image and non-image data, with the option to include a custom logo within the QR code. This report outlines the system's design, implementation, and features, highlighting the use of Streamlit for building interactive and user-friendly web applications.

**Introduction**

QR codes have become a widely used tool for encoding information in a compact and accessible format. They can store various types of data, including URLs, text, and images, making them versatile for different applications. This project aims to create a QR-Code generator using Python and Streamlit, providing an interactive interface for users to generate customized QR codes. The application allows users to input data, select options for including images or logos, and generate QR codes with ease.

**Problem Statement**

Traditional QR code generation tools often lack flexibility and customization options. Users may need a more intuitive interface to generate QR codes with specific features, such as embedded logos or custom data. The challenge is to develop a user-friendly application that offers robust QR code generation capabilities.

**Key Challenges:**

1. **Flexibility**: Provide options for generating QR codes from both images and non-image data.
2. **Customization**: Allow users to add logos or custom images within the QR code.
3. **User Interface**: Develop an intuitive and easy-to-use interface for users to input data and generate QR codes.

**System Design**

The design of the QR-Code generator application focuses on providing a seamless user experience while ensuring flexibility and customization options. The system is composed of several components, each responsible for different functionalities.

**User Interface**

The user interface is designed to be intuitive and user-friendly, featuring:

* **Text Input Fields**: For entering the name of the QR code and the content to be encoded.
* **Checkboxes**: For selecting options to generate QR codes for images, non-image data, and whether to include a background logo.
* **File Uploaders**: For uploading image files to be included in the QR code.
* **Buttons**: For triggering the QR code generation process.

**QR Code Generation**

The QR code generation component handles the creation of QR codes based on the user inputs:

* **Data Handling**: Manages the data input by users, including text and images.
* **QR Code Creation**: Uses the qrcode library to generate QR codes from the provided data.
* **Image Processing**: Utilizes the PIL library to include logos within the QR code if specified.

**Security Measures**

The application ensures the secure handling of user data:

* **Data Validation**: Validates user inputs to prevent errors during QR code generation.
* **File Handling**: Manages uploaded files securely within the application session.

**Implementation**

The implementation of the QR-Code generator application is detailed below:

**Code Implementation**

import streamlit as st

from streamlit\_multi\_menu import streamlit\_multi\_menu

import qrcode

from PIL import Image

import time

st.title("QR-Code Generator")

column1, column2 = st.columns(2)

check1 = None

check2 = None

image = None

check3 = False

col3, col4 = st.columns(2)

name = st.text\_input('Name for Your QR')

with column1:

check1 = st.checkbox('Generate QR for Image')

with column2:

check2 = st.checkbox('Generate QR for Non Image')

with col3:

check3 = st.checkbox('You need background?')

if check1:

image\_data = st.file\_uploader('Give your Image to QR', ['png', 'jpg'])

if check3:

image = st.file\_uploader('Give your logo', ['png', 'jpg'])

if check2:

data = st.text\_input("Give your content that need to be in a QR")

button = st.button('Generate QR')

class QRCODE\_generator():

def \_\_init\_\_(self, data, name, background\_color='black'):

self.data = data

self.image = None

self.name = name

self.background\_color = background\_color

def with\_logo\_generate(self, image):

self.image = image

qr = qrcode.QRCode(version=3, box\_size=10, border=5)

passed\_data = self.data

qr.add\_data(passed\_data)

qr.make(fit=True)

if self.image:

try:

img = qr.make\_image(fill\_color=self.background\_color, back\_color='white')

logo = Image.open(self.image)

logo = logo.resize((50, 50))

img\_w, img\_h = img.size

logo\_w, logo\_h = logo.size

pos = ((img\_w - logo\_w) // 2, (img\_h - logo\_h) // 2)

img.paste(logo, pos)

img.save(f"storage/{self.name}\_with\_logo.png")

except:

pass

def generate(self):

qr = qrcode.QRCode(version=3, box\_size=10, border=5)

passed\_data = self.data

qr.add\_data(passed\_data)

qr.make(fit=True)

img = qr.make\_image(fill\_color=self.background\_color, back\_color="white")

img.save(f"storage/{self.name}.png")

if button:

if check1 and not check3:

image\_data1 = Image.open(image\_data)

qr = QRCODE\_generator(image\_data1, name)

with st.spinner('Wait for it...'):

time.sleep(5)

st.success("Done!")

qr.generate()

st.image(f"storage/{name}.png")

if check3 and check1:

image\_data1 = Image.open(image\_data)

qr = QRCODE\_generator(image\_data1, name)

with st.spinner('Wait for it...'):

time.sleep(5)

st.success("Done!")

qr.with\_logo\_generate(image)

st.image(f"storage/{name}\_with\_logo.png")

if check3 and check2:

qr = QRCODE\_generator(data, name)

with st.spinner('Wait for it...'):

time.sleep(5)

st.success("Done!")

qr.with\_logo\_generate(image)

st.image(f"storage/{name}\_with\_logo.png")

if check2 and not check3:

qr = QRCODE\_generator(data, name)

with st.spinner('Wait for it...'):

time.sleep(5)

st.success("Done!")

qr.generate()

st.image(f'storage/{name}.png')

**Input Design**

* **Text Inputs**: Users provide a name for their QR code and the content to be encoded.
* **Checkboxes**: Users select options for generating QR codes for images, non-image data, and whether to include a background logo.
* **File Uploaders**: Users upload image files to be included in the QR code.

**Output Design**

* **Generated QR Codes**: Display the generated QR codes with or without logos, based on user selections.
* **Status Messages**: Inform users of the progress and completion of the QR code generation process.

**Testing and Validation**

Testing is crucial to ensure the application functions correctly and meets user expectations. Several test scenarios were conducted to validate the QR-Code generator's functionality.

**Test Scenarios**

1. **Generate QR Code for Image**:
   * **Input**: Image file and name.
   * **Expected Output**: QR code image generated successfully.
2. **Generate QR Code with Logo**:
   * **Input**: Image file, logo file, and name.
   * **Expected Output**: QR code image with embedded logo generated successfully.
3. **Generate QR Code for Non-Image Data**:
   * **Input**: Text data and name.
   * **Expected Output**: QR code image generated successfully.
4. **Generate QR Code with Logo for Non-Image Data**:
   * **Input**: Text data, logo file, and name.
   * **Expected Output**: QR code image with embedded logo generated successfully.

**Test Results**

The test scenarios confirmed that the application performs as expected. The QR codes were generated correctly based on user inputs, and the status messages provided appropriate feedback during the process.

**Security Considerations**

The application ensures the secure handling of user data through several measures:

* **Data Validation**: Validates user inputs to prevent errors during QR code generation.
* **File Handling**: Manages uploaded files securely within the application session, preventing unauthorized access.

**Future Enhancements**

Several enhancements can be made to improve the application:

1. **Database Integration**: Store generated QR codes and user data in a database for future reference and management.
2. **Custom Colors**: Allow users to choose custom colors for the QR code.
3. **Enhanced Error Handling**: Implement more robust error handling to manage unexpected issues gracefully.
4. **Advanced QR Code Features**: Include options for generating QR codes with different shapes, patterns, and additional embedded information.

**Conclusion**

The QR-Code generator application developed using Python and Streamlit provides a user-friendly and flexible solution for generating QR codes. The system design ensures an intuitive interface, customization options, and secure handling of user data. This project demonstrates the practical use of Python and Streamlit in developing interactive web applications, offering a foundation for further enhancements and exploration in QR code generation.