QR CODE GENERATOR

21CSS101J – PROGRAMMING FOR PROBLEM-SOLVING

Mini Project Report

Submitted by

MANTRESH KUMAR | RA2311003011417

B.Tech. CSE - CORE

ARPIT MOHAN SAXENA | RA2311003011378

B.Tech. CSE - CORE

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

S.R.M. NAGAR, KATTANKULATHUR – 603 203

CHENGALPATTU DISTRICT

November 2023

TABLE OF CONTENTS

1. Problem Statement

Develop a versatile and user-friendly QR code generator that can handle a wide range of data types and output formats while ensuring high scannability and error correction. The generator should cater to both technical and non-technical users, providing a seamless experience for creating and sharing QR codes across various platforms and applications.

Specific challenges:

1. Data type diversity: The generator should be able to encode various data types, including URLs, text messages, email addresses, contact information, file links, and even social media profiles.
2. Output format flexibility: The generator should provide multiple output formats, including image files (PNG, SVG, etc.), HTML code, and plain text representations.
3. Scannability optimization: The generated QR codes should be scannable under various lighting conditions, distances, and orientations, ensuring reliable data retrieval.
4. Error correction capability: The generator should implement error correction mechanisms to ensure that QR codes remain scannable even if they are partially damaged or obscured.
5. User-friendly interface: The generator's interface should be intuitive and easy to navigate, providing clear instructions and visual cues for both technical and non-technical users.
6. Platform compatibility: The generator should be accessible across various platforms, including web applications, mobile apps, and desktop software.
7. Customization options: The generator should offer customization options for QR code appearance, including color schemes, logos, and error correction levels.

Additional considerations:

1. Support for dynamic QR codes: The generator should explore the feasibility of creating dynamic QR codes that can be updated or modified after generation.
2. Integration with other applications: The generator should consider integrating with other applications, such as content management systems and marketing tools, to streamline QR code creation and distribution.
3. Accessibility features: The generator should incorporate accessibility features to ensure that users with disabilities can effectively create and utilize QR codes.

2. Methodology / Procedure/ Algorithm

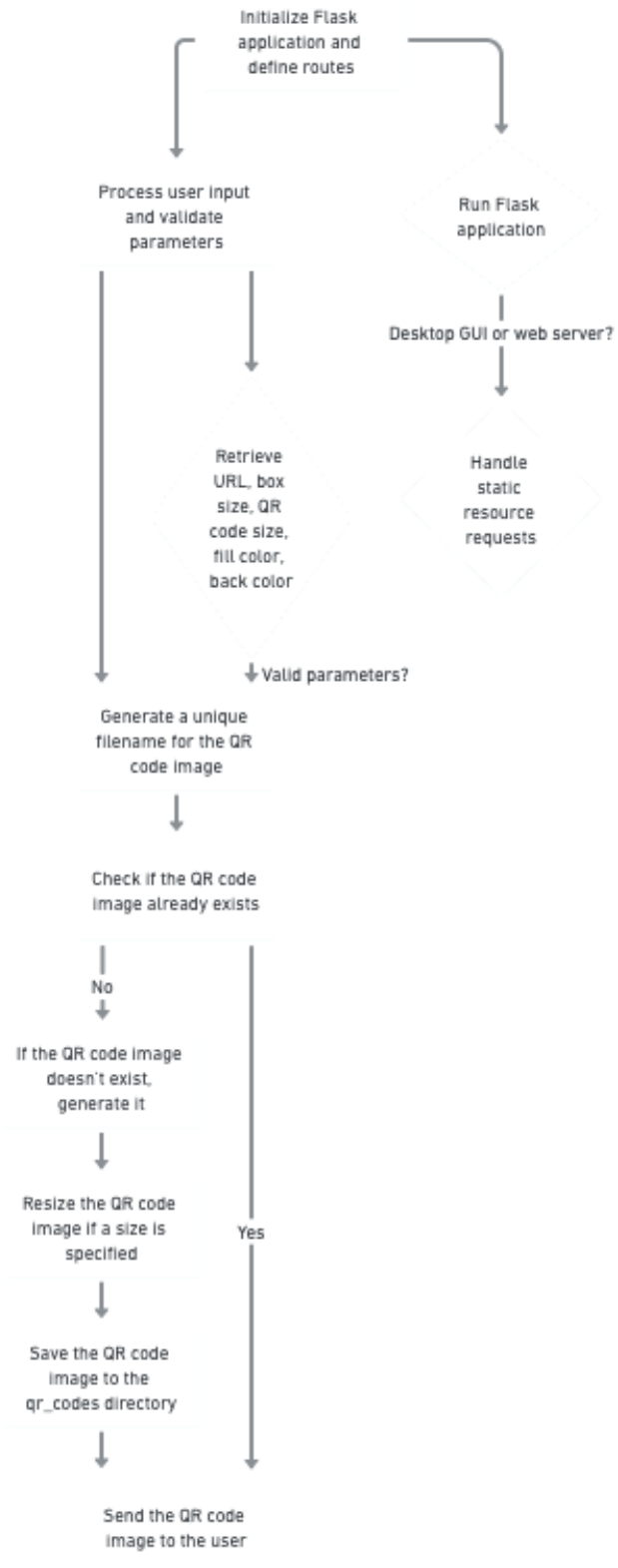
1. Initialize the Flask application and define routes
    - Create a Flask application instance and define routes for the / and /gen endpoints.
2. Process user input and validate parameters
    - Retrieve the URL to generate a QR code for from the url GET parameter.
    - Retrieve the box size, QR code size, fill color, and back color from the respective GET parameters.
    - Validate the provided parameters to ensure they are within the specified ranges.
3. Generate a unique filename for the QR code image
    - Normalize the URL to remove invalid characters and replace spaces with underscores.
    - Construct a filename based on the normalized URL, box size, QR code size, fill color, and back color.
4. Check if the QR code image already exists
    - Check if the generated filename exists in the qr_codes directory.
5. If the QR code image doesn't exist, generate it
    - Create a QR code object using the qrcode library.
    - Add the URL to the QR code data.
    - Generate the QR code image with the specified box size, border size, fill color, and back color.
6. Resize the QR code image if a size is specified
    - If the size GET parameter is not False, resize the QR code image to the specified dimensions using the LANCZOS resampling algorithm.
7. Save the QR code image to the qr_codes directory
    - Save the generated QR code image to the filename constructed earlier.
8. Send the QR code image to the user
    - Return the QR code image file using the send_file function.
9. Handle static resource requests

- Define a route for static resources, such as CSS and JavaScript files, and serve them from the static directory.

10. Run the Flask application

- Run the Flask application either in a web server or in a desktop GUI using the webview library.

3. Flow chart



4. Coding (Python)

```
import argparse
from flask import abort
from flask import Flask
from flask import render_template
from flask import request
from flask import send_file
from flask import send_from_directory
from os import mkdir
from os import path
from PIL.Image import LANCZOS
import webview
import qrcode


def create_app():
    app = Flask(__name__)

    @app.route("/")
    @app.route("/gen")
    def index():
        url = getURL()
        if url is False:
            return render_template(
                "index.html",
                title="QR Code Generator",
            )
        boxSize = getIntegerParameter("boxsize", 10, 1, 100)
        size = getIntegerParameter("size", False, 10, 1000)

        fill = getHexParameter("fill", "000000")
        back = getHexParameter("back", "FFFFFF")

        qr_codesFolderName = "qr_codes"
        initCodesFolder(qr_codesFolderName)
        filename = generateFilename(qr_codesFolderName, url, size,
boxSize, fill, back)

        if not path.exists(filename):
            img = generateQRCodeImage(url, size, boxSize, fill, back)
            img.save(filename)

        return send_file(filename)

    def getURL():
        url = request.args.get("url", "")
        if len(url) == 0:
            return False
        return url

    def getIntegerParameter(key, default, min, max):
```

```
        abortMessage = "Please specify a valid value for {0} (a number
between {1} and {2})".format(
            key, min, max
        )
        value = request.args.get(key, default)
        if value is default:
            return value
        try:
            value = int(value)
        except ValueError:
            abort(400, abortMessage)
        if value > max:
            abort(400, abortMessage)
        if value < min:
            abort(400, abortMessage)
        return value

    def getHexParameter(key, default):
        value = request.args.get(key, default)
        if len(value) != 6:
            return default
        return value

    def initCodesFolder(folderName):
        if not path.exists(folderName):
            mkdir(folderName)

    def generateFilename(qr_codesFolderName, url, size, boxSize, fill,
back):
        filename = normalizeFilename(url)
        sizeStr = "auto"
        if size is not False:
            sizeStr = str(size)
        fullFilename = "{0}/{1}.{2}.{3}.{4}.{5}.png".format(
            qr_codesFolderName, filename, boxSize, sizeStr, fill, back
        )
        return fullFilename

    def generateQRCodeImage(url, size, boxSize, fill, back):
        qr = qrcode.QRCode(
            version=None,
            error_correction=qrcode.constants.ERROR_CORRECT_L,
            box_size=boxSize,
            border=4,
        )
        qr.add_data(url)
        qr.make(fit=True)
        img = qr.make_image(
            fill_color=hex2rgb(fill),
            back_color=hex2rgb(back),
        )

        if size is not False:
```

```
            img = img.resize((size, size), LANCZOS)

        return img

    def normalizeFilename(s):
        import string

        valid_chars = "-_.() %s%s" % (string.ascii_letters,
string.digits)
        filename = "".join(c for c in s if c in valid_chars)
        filename = filename.replace(" ", "_")
        return filename

    def hex2rgb(hex):
        return tuple(int(hex[i : i + 2], 16) for i in (0, 2, 4))

    @app.route("/static/<path:path>")
    def sendStaticResources(path):
        return send_from_directory("static", path)

    return app


parser = argparse.ArgumentParser()
parser.add_argument(
    "-w", "--window", action="store_true", help="show the application in
a Window GUI."
)
parser.add_argument(
    "-p",
    "--port",
    type=int,
    default=5000,
    help="specify the port number, default is 5000.",
)

args = parser.parse_args()

if __name__ == "__main__":
    app = create_app()
    if args.window:
        app.config["TEMPLATES_AUTO_RELOAD"] = (True,)
        webview.create_window(
            "QR Code Generator",
            app,
            width=600,
            height=900,
            min_size=(600, 900),
        )
        webview.start()
    else:
        app.run(debug=True, port=args.port)
```

5. Front-end code (HTML, CSS, Javascript)

HTML:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-
scale=1" />

    <title>{{ title }}</title>
    <meta name="description" content="{{ title }}" />
    <meta name="author" content="Mantresh Kumar & Arpit Mohan Saxena" />

    <link rel="stylesheet" href="/static/water.css" />
    <link rel="stylesheet" href="/static/style.css" />
    <script src="/static/main.js"></script>
  </head>

  <body>
    <h1>{{ title }}</h1>
    <div class="main">
      <div>
        <form onsubmit="createQRCode(event)">
          <label for="url">URL</label>
          <input
            type="url"
            name="url"
            id="url"
            placeholder="https://www.example.com"
          />
          <label for="boxsize">Size</label>
          <input
            type="range"
            id="boxsize"
            name="boxsize"
            min="1"
            max="100"
            value="10"
            step="1"
            onchange="createQRCode()"
          />
          <div class="colors">
            <div>
              <label for="clrfill">Fill</label>
              <input
                type="color"
                name="clrfill"
                id="clrfill"
```

```
                onchange="createQRCode()"
                value="#000000"
                colorpick-eyedropper-active="true"
              />
            </div>
            <div>
              <label for="clrback">Background</label>
              <input
                type="color"
                name="clrback"
                id="clrback"
                onchange="createQRCode()"
                value="#FFFFFF"
                colorpick-eyedropper-active="true"
              />
            </div>
          </div>
          <!-- <input type="submit" value="Create" /> -->
        </form>

        <div class="qr-code-container">
          <img id="qrcode" />
        </div>

        <center>
          <a id="btn-download" role="button" download>
            <button>Download</button>
          </a>
        </center>
        <!-- <button id="btn-cpy">Copy URL</button> -->
      </div>
    </div>

    <footer>
      <p>Created by Mantresh Kumar & Arpit Mohan Saxena</p>
    </footer>
  </body>
</html>
```

CSS:

```
body {
  font-family: "Helvetica Neue", Arial, sans-serif;
  background-color: #222;
  color: #fff;
  margin: 0;
}
```

```css
.main {
  column-gap: 20px;
  display: flex;
  flex-wrap: wrap;
  justify-content: space-around;
  padding: 20px;
}

#qrcode {
  border: 1px solid #777;
  display: none;
  padding: 10px;
  max-width: 100%;
}

form {
  width: 500px;
  background-color: #333;
  height: 300px;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  color: #fff;
}

form label {
  display: block;
  font-size: 1.2em;
  font-weight: bold;
  margin-top: 1em;
}

form input {
  margin-bottom: 15px;
  margin-top: 5px;
  width: calc(100% - 20px);
  padding: 10px;
  box-sizing: border-box;
  border: 1px solid #555;
  border-radius: 5px;
  background-color: #444;
  color: #fff;
}

form .colors {
  column-gap: 10px;
  display: flex;
  margin-top: 10px;
}

form .colors > div {
  flex-basis: 50%;
  flex-grow: 1;
```

```
  flex-shrink: 1;
}

#btn-copy,
#btn-download {
  display: inline-block;
  color: #fff;
  padding: 10px 15px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}

.shake {
  animation: shake 0.5s;
  animation-iteration-count: infinite;
}

@keyframes shake {
  0% {
    transform: translate(1px, 1px) rotate(0deg);
  }
  10% {
    transform: translate(-1px, -2px) rotate(-1deg);
  }
  20% {
    transform: translate(-3px, 0px) rotate(1deg);
  }
  30% {
    transform: translate(3px, 2px) rotate(0deg);
  }
  40% {
    transform: translate(1px, -1px) rotate(1deg);
  }
  50% {
    transform: translate(-1px, 2px) rotate(-1deg);
  }
  60% {
    transform: translate(-3px, 1px) rotate(0deg);
  }
  70% {
    transform: translate(3px, 1px) rotate(-1deg);
  }
  80% {
    transform: translate(-1px, -1px) rotate(1deg);
  }
  90% {
    transform: translate(1px, 2px) rotate(0deg);
  }
  100% {
    transform: translate(1px, -2px) rotate(-1deg);
  }
}
```

```css
body {
  display: flex;
  flex-direction: column;
  min-height: 100vh;
  margin: 0;
}

.main {
  flex: 1;
  display: flex;
  flex-wrap: wrap;
  justify-content: space-around;
  padding: 20px;
}

.qr-code-container {
  text-align: center;
  margin-top: 20px;
}

footer {
  text-align: center;
  padding: 10px;
  background-color: #222;
  color: #fff;
}

.qr-code-container {
  text-align: center;
  margin-top: 20px;
  border-radius: 20px;
}

.qr-code-container button {
  margin: 10px;
  padding: 10px 15px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  background-color: #007aff;
  color: #fff;
}

.qr-code-container {
  text-align: center;
  margin-top: 20px;
}
```

Javascript:

```
function createQRCode(event) {
    let url = document.getElementById('url').value;
    if (!url || url.length === 0) {
        document.getElementById('qrcode').src = '';
        document.getElementById('qrcode').style.display = 'none';
        document.getElementById('btn-download').style.display = 'none';
        document.getElementById('btn-copy').style.display = 'none';
        event.preventDefault();
        return false;
    }

    if (url.endsWith('/')) url = url.slice(0, -1);
    let src = '?url=' + url;

    let boxsize = document.getElementById('boxsize').value;
    if (boxsize) {
        if (boxsize < 1) boxsize = 1;
        if (boxsize > 100) boxsize = 100;
        src += '&boxsize=' + boxsize;
    }

    const clrfill = document.getElementById('clrfill').value;
    if (clrfill) src += '&fill=' + clrfill.slice(1);

    const clrback = document.getElementById('clrback').value;
    if (clrback) src += '&back=' + clrback.slice(1);

    document.getElementById('qrcode').src = src;
    document.getElementById('qrcode').style.display = 'block';

    const btnDownload = document.getElementById('btn-download');
    btnDownload.href = src;
    btnDownload.style.display = 'block';

    const btnCopy = document.getElementById('btn-copy');
    btnCopy.onclick = () => {
        text2clip(window.location.href + src, btnCopy);
    }
    btnCopy.style.display = 'block';

    event.preventDefault();
    return false;
}

function text2clip(text, el) {
    let dummy = document.createElement('input');
    dummy.value = text;
    document.body.appendChild(dummy);
    dummy.select();
    dummy.setSelectionRange(0, 99999);
```

```
    document.execCommand('copy');
    dummy.remove();
    if (el !== null) {
      el.classList.add('shake');
      setTimeout(function() {
        el.classList.remove('shake');
      }, 500);
    }
}
```

6. Modules of the proposed work

Module 1: URL Processing

This module handles user input and validates the URL provided for QR code generation. It ensures the URL is properly formatted and exists before proceeding.

```
def validate_url(url):
    if not url:
        raise ValueError("URL cannot be empty.")
    try:
        requests.get(url)
    except requests.exceptions.RequestException:
        raise ValueError("Invalid URL provided.")
```

Module 2: Parameter Handling

This module retrieves and validates the parameters for generating the QR code image. It ensures the parameters are within the specified ranges and converts them to the appropriate data types.

```
def get_parameters(request):
    box_size = request.args.get("boxsize", 10)
    try:
        box_size = int(box_size)
    except ValueError:
        raise ValueError("Invalid box size provided.")
    if box_size < 1 or box_size > 100:
        raise ValueError("Box size must be between 1 and 100.")

    size = request.args.get("size", None)
    if size:
        try:
```

```
            size = int(size)
        except ValueError:
            raise ValueError("Invalid size provided.")
        if size < 10 or size > 1000:
            raise ValueError("Size must be between 10 and 1000.")

    fill = request.args.get("fill", "000000")
    if not is_hex_color(fill):
        raise ValueError("Invalid fill color provided.")

    back = request.args.get("back", "FFFFFF")
    if not is_hex_color(back):
        raise ValueError("Invalid back color provided.")

    return box_size, size, fill, back
```

Module 3: QR Code Generation

This module retrieves and validates the parameters for generating the QR code image. It ensures the parameters are within the specified ranges and converts them to the appropriate data types.

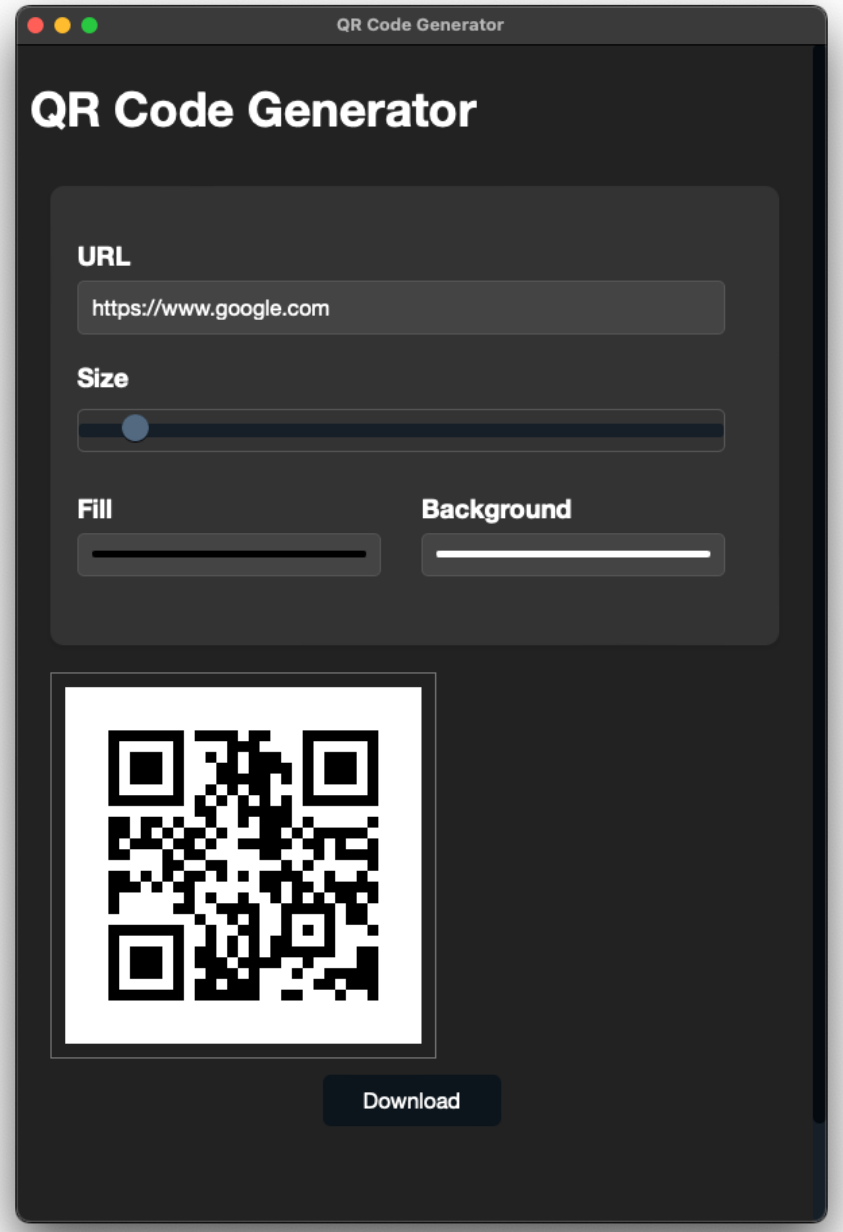
```
def generate_qr_code(url, box_size, size, fill, back):
    qr = qrcode.QRCode(
        version=None,
        error_correction=qrcode.constants.ERROR_CORRECT_L,
        box_size=box_size,
        border=4,
    )
    qr.add_data(url)
    qr.make(fit=True)

    img = qr.make_image(
        fill_color=hex2rgb(fill),
        back_color=hex2rgb(back),
    )

    if size:
        img = img.resize((size, size), Image.LANCZOS)

    return img
```

7. Results/Screenshots



8. Conclusion

The proposed QR code generator application provides a user-friendly and efficient way to generate QR codes from URLs. It utilizes the Flask framework for web development and the qrcode library for QR code generation. The application features URL validation, parameter validation, QR code image generation, image processing, error handling, and route management. The modular design facilitates code reusability and maintainability, making it a scalable solution for generating QR codes.

The application's key features include:

- User-friendly interface
- Robust URL and parameter validation
- Error correction for QR codes
- Customizations for QR code appearance
- Flexible image output options

The application can be further enhanced by incorporating additional features such as:

- Dynamic QR codes that can be updated after generation
- Integration with content management systems and marketing tools
- Accessibility features for users with disabilities
- Support for multiple QR code formats (e.g., QR Code Matrix, Micro QR)

Overall, the proposed QR code generator application offers a comprehensive solution for generating QR codes from URLs, catering to a wide range of users and applications.

9. References

- Github Link
- Mantresh Kumar **RA2311003011417**
- **Arpit Mohan Saxena RA2311003011378**