project : QR Code Generation .

It allows users to generate a QR code with details, scan it to view the information, and process it through an API call.

1)Generating a QR Code:

A user provides their name or any details.

The system generates a QR code containing this information.

The QR code is saved as an image in the system.

2)Scanning the QR Code:

When scanned using a mobile camera or QR scanner app, it displays the details it just shows the details.

\*Added Dependency’s:-

1)Spring Boot Starter

Provides core Spring Boot functionalities.

2)Spring Boot Starter Web

Enables web application development with Spring MVC.

3)Spring Boot Starter Data JPA

java Persistence API (JPA)

4)ZXing Core (QR Code Generation Library)

Core library for QR code generation.

5)ZXing JavaSE (QR Code Generation Library)

Java SE implementation for generating QR codes.

6)Oracle JDBC Driver

Enables connectivity with an Oracle database.

Project Structure:-  
 QrGeneration/

│── src/

│ |---main/

│ │ |---java/com/qrcode/QrGeneration/

│ │ │ |---controller/

│ │ │ │ |-─ QrCodeController.java

│ │ │ |-─ model/

│ │ │ │ |---QRCodeEntity.java

│ │ │ |---repository/

│ │ │ │ |---QRCodeRepository.java

│ │ │ |--- service/

│ │ │ │ |---QRCodeService.java

│ │ |---resources/

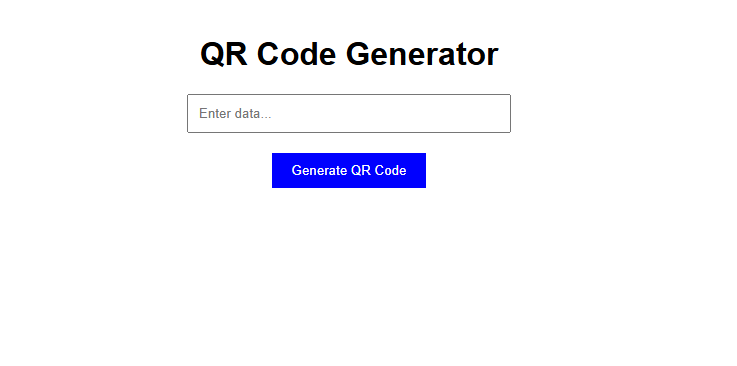
│ │ │ |---application.properties

│ │ |---resources/static

| | | |----index.htlml

│── pom.xml

│── QrGenerationApplication.java

Output:- http://localhost:8080/index.html  


Here, enter some data in this text box, and that data will be converted into a QR code.  
 Text data is received as input.

A unique filename is generated for the QR Code image.

The folder structure is created if it doesn’t exist.

The data is encoded into a QR Code format using QRCodeWriter.encode().

The QR Code is saved as a .png file using MatrixToImageWriter.writeToPath().

The QR Code data is stored in the database.

The generated QR Code filename is returned as output

Here,  
The .png file is saved at the following path: C:\\Users\\hr390\\Desktop\\qrcodes.



->Enter some data in the text box and click on “Generate QR code” button

->To view the generated QRcode

->click on “Display QR” button

Frontend: html/css /js

Placing the frontend (index.html) inside ”src/main/resources/static/ “ in a **Spring Boot project.**

-> Spring Boot automatically serves static files from src/main/resources/static/.

-> No need for an external web server—just run the Spring Boot app, and it serves index.html directly.  
  
Backend technologies :

**Java Spring Boot** – Main backend framework  
 **Oracle Database** – For storing QR code details  
 **JPA (Java Persistence API)** – For database interactions  
 **Feign Client (in some projects)** – For service-to-service communication  
 **Google ZXing Library** – For QR code generation

Implementation :  
  
1.Model Class (Database Table)

Stores details

package com.qrcode.QrGeneration.model;

import jakarta.persistence.\*;

@Entity

@Table(name = "qr\_codes")

public class QRCodeEntity {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false, columnDefinition = "CLOB")

private String data;

@Column(nullable = false)

private String qrCodePath;

// Getters and Setters

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getData() {

return data;

}

public void setData(String data) {

this.data = data;

}

public String getQrCodePath() {

return qrCodePath;

}

public void setQrCodePath(String qrCodePath) {

this.qrCodePath = qrCodePath;

}

}

2. QR Code Generation(service) :

Converts details into a QR code image.

package com.qrcode.QrGeneration.service;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import org.springframework.stereotype.Service;

import com.qrcode.QrGeneration.model.QRCodeEntity;

import com.qrcode.QrGeneration.repository.QRCodeRepository;

import com.google.zxing.BarcodeFormat;

import com.google.zxing.WriterException;

import com.google.zxing.client.j2se.MatrixToImageWriter;

import com.google.zxing.common.BitMatrix;

import com.google.zxing.qrcode.QRCodeWriter;

@Service

public class QRCodeService {

private final QRCodeRepository qrCodeRepository;

public QRCodeService(QRCodeRepository qrCodeRepository) {

this.qrCodeRepository = qrCodeRepository;

}

public String generateQRCode(String data) {

try {

// Generate unique filename

String timestamp = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyyMMddHHmmss"));

String fileName = "qrcode\_" + timestamp + ".png";

String folderPath = "C:\\Users\\hr390\\Desktop\\qrcodes";

Path directory = Paths.get(folderPath);

// Create directory if not exists

if (!Files.exists(directory)) {

Files.createDirectories(directory);

}

Path filePath = directory.resolve(fileName);

int width = 300;

int height = 300;

QRCodeWriter qrCodeWriter = new QRCodeWriter();

BitMatrix bitMatrix = qrCodeWriter.encode(data, BarcodeFormat.QR\_CODE, width, height);

MatrixToImageWriter.writeToPath(bitMatrix, "PNG", filePath);

// Save data in database

QRCodeEntity qrCode = new QRCodeEntity();

qrCode.setData(data);

qrCode.setQrCodePath("qrcodes/" + fileName); // Save relative path

qrCodeRepository.save(qrCode);

return fileName;

} catch (WriterException | IOException e) {

e.printStackTrace();

return "Error generating QR Code";

}

}

}

3. REST API Endpoints (Controller) :

@RequestMapping("/api/qrcode")

/generate → Generates a QR code .

package com.qrcode.QrGeneration.controller;

import org.springframework.web.bind.annotation.\*;

import com.qrcode.QrGeneration.service.QRCodeService;

import com.qrcode.QrGeneration.model.QRCodeEntity;

@RestController

@RequestMapping("/api/qrcode")

public class QRCodeController {

private final QRCodeService qrCodeService;

public QRCodeController(QRCodeService qrCodeService) {

this.qrCodeService = qrCodeService;

}

@PostMapping("/generate")

public String generateQRCode(@RequestBody QRCodeEntity qrCodeEntity) {

if (qrCodeEntity.getData() == null || qrCodeEntity.getData().isEmpty()) {

return "Invalid request: 'data' field is required";

}

return qrCodeService.generateQRCode(qrCodeEntity.getData());

}

}

4. Repository:

To perform operations in DB.

package com.qrcode.QrGeneration.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.qrcode.QrGeneration.model.QRCodeEntity;

public interface QRCodeRepository extends JpaRepository<QRCodeEntity, Long> {

}

5) application.properties :

spring.application.name=QrGeneration

server.port=8080

# Oracle Database Configuration

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:XE

spring.datasource.username=system

spring.datasource.password=oracle1

spring.datasource.driver-class-name=oracle.jdbc.OracleDriver

# Hibernate JPA Properties

spring.jpa.database-platform=org.hibernate.dialect.OracleDialect

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

# Enable Static Resource Mapping

spring.web.resources.static-locations=classpath:/static/,file:///C:/Users/hr390/Desktop/qrcodes/

Frontend code :

<!**DOCTYPE** html>

<**html** lang=*"en"*>

<**head**>

<**meta** charset=*"UTF-8"*>

<**meta** name=*"viewport"* content=*"width=device-width, initial-scale=1.0"*>

<**title**>QR Code Generator</**title**>

<**style**>

body {

font-family: *Arial, sans-serif*;

text-align: *center*;

margin-top: *50px*;

}

input {

padding: *10px*;

width: *300px*;

margin-bottom: *10px*;

}

button {

padding: *10px 20px*;

cursor: *pointer*;

background: *blue*;

color: *white*;

border: *none*;

margin-top: *10px*;

}

button:disabled {

background: *gray*;

cursor: *not-allowed*;

}

img {

margin-top: *20px*;

width: *300px*;

height: *300px*;

border: *1px solid #ddd*;

}

</**style**>

</**head**>

<**body**>

<**h1**>QR Code Generator</**h1**>

<**input** type=*"text"* id=*"qrData"* placeholder=*"Enter data..."*>

<**br**>

<**button** onclick="generateQRCode()">Generate QR Code</**button**>

<**br**>

<!-- Initially hidden display button -->

<**button** id=*"displayBtn"* onclick="displayQRCode()" style="display: *none*;">Display QR Code</**button**>

<**div** id=*"qrCodeContainer"*></**div**>

<**script**>

let qrCodeFilePath = "";

**function** generateQRCode() {

**const** data = document.getElementById("qrData").value;

**if** (!data) {

alert("Please enter some data.");

**return**;

}

fetch("http://localhost:8080/api/qrcode/generate", {

method: "POST",

headers: {

"Content-Type": "application/json"

},

body: JSON.stringify({ data: data })

})

.then(response => response.text())

.then(filePath => {

qrCodeFilePath = filePath;

document.getElementById("displayBtn").style.display = "inline-block"; // Show Display button

document.getElementById("qrCodeContainer").innerHTML = ""; // Clear previous QR code

})

.**catch**(error => alert("Failed to generate QR Code."));

}

**function** displayQRCode() {

**if** (qrCodeFilePath) {

document.getElementById("qrCodeContainer").innerHTML = `

<h3>Generated QR Code:</h3>

<img src="http://localhost:8080/${qrCodeFilePath}" alt="QR Code">`;

}

}

</**script**>

</**body**>

</**html**>