PROPOSAL

📄 Project Title: QR-Based URL Safety Checker (QR Safety App)

🔍 Introduction

With the widespread use of **QR codes** in modern digital transactions, advertisements, and access systems, their convenience also brings potential **security risks**. Most devices instantly redirect to a URL when a QR code is scanned — often **without verifying** whether the destination is safe or malicious.

While URL safety checkers exist for manually entered URLs, there is currently **no mainstream web-based solution** that verifies **QR-embedded URLs** before accessing them. This project addresses that gap.

🎯 Objective

To develop a **Progressive Web App (PWA)** that:

- Scans a QR code using the device camera
- Extracts and displays the encoded URL
- Automatically checks if the URL is **safe or malicious** using the **Google Safe Browsing (GSB) API**
- Presents the user with a clear safety status, **without redirecting them to the URL**

🔒 Problem Statement

QR codes:

- Can embed **phishing**, **malware**, or **malicious redirection** links
- Often auto-open in the browser with **no intermediate validation**
- Are used in public spaces where users may **scan unknowingly**

This project introduces a **pre-check layer** that prioritizes **user awareness and safety**.

🧠 Solution Overview

The system is a **mobile-friendly web application** built using:

- **HTML, CSS, JavaScript** for the frontend
- **Flask (Python)** for backend processing
- **Google Safe Browsing API** for threat analysis

Additionally, the application is configured as a **Progressive Web App (PWA)**, enabling users to:

- Install it like a mobile or desktop app
- Access it offline (optional)
- Use it in full-screen app-like mode

🧰 Technology Stack

Component	Technology Used
Frontend UI	HTML5, CSS3, JavaScript
QR Code Scanning	`html5-qrcode` JavaScript library
Backend API	Python (Flask Framework)
URL Safety Checker	Google Safe Browsing API
PWA Integration	Web App Manifest + Service Worker (optional)
Deployment	Netlify (frontend) + Render/PythonAnywhere (backend)

🔁 Workflow

1. **User Interface**
   - User opens the web app (mobile or desktop)
   - Clicks a button to scan a QR code using the device's camera
2. **QR Code Processing**
   - The app uses `html5-qrcode` to scan the QR code
   - Extracts the data (assumed to be a URL)
3. **URL Safety Verification**
   - The extracted URL is sent to the backend (Flask)
   - The backend queries the **Google Safe Browsing API**
   - The response is analyzed to determine if the URL is safe
4. **Result Display**
   - The frontend shows the URL and a visual indicator:
     - ✅ Safe
     - ⚠️ Suspicious
     - ❌ Malicious
5. **PWA Behavior**
   - The app includes a manifest (`manifest.json`) for installability
   - When accessed on supported browsers, users can **install the app** to their home screen or desktop
   - Once installed, it launches in **full-screen**, app-like mode

---

🌟 Key Features

- **Cross-platform**: Works on both desktop and mobile
- **Camera integration**: Enables QR scanning without needing external apps
- **Real-time threat check**: Uses a trusted source (Google) to ensure security
- **User control**: Does not auto-open any link — puts safety and choice in user's hands
- **Installable**: Functions like a native app via PWA capabilities

---

📌 Future Enhancements (Optional Research Directions)

- Add support for **manual URL entry**
- Provide **threat type classification** (phishing, malware, etc.)
- Implement **offline history** of scanned and verified links
- Add **multi-language support**
- Improve UI with frameworks like **React** or **Tailwind CSS**

✅ Conclusion

This project provides a timely solution to a rising concern in digital safety: **blindly scanning and opening QR codes without verification**. By leveraging modern web technologies and external threat detection APIs, it delivers a secure, fast, and installable solution that raises awareness and protects users.

With minimal resource requirements and scalable design, this tool could be adopted in educational institutions, public venues, and secure environments as a **first-layer defense against QR-based threats**.

END