PROJECT ABSTRACT

QR Code Generator and Reader Desktop App

A desktop utility that generates standards-compliant QR codes (URL, text, Wi-Fi, vCard, UPI) with custom colours, logo embedding and batch mode, and reads codes back from screen captures or image files, with a persistent local history. Delivered built-to-order with report, PPT and Q&A.;

01. Title

QR Code Generator and Reader Desktop App

02. Introduction / Background

QR codes have become the default bridge between physical and digital: Wi-Fi sharing without dictating passwords, UPI payment counters, event badges, contact exchange. The QR standard (ISO/IEC 18004) packs this data into a matrix of modules protected by Reed–Solomon error correction, so a code stays readable even when partly obscured — the property that makes centre-logo embedding possible. Free web generators cover the basics but watermark output, upload payload data to a server, and cannot read a code back. A desktop utility removes all three problems: generation and decoding run fully offline, there are no watermarks, and one tool handles both directions. QRDesk is built on Python’s qrcode/Pillow stack for encoding and the ZBar decoding engine (via pyzbar) for reading, wrapped in a native Tkinter interface.

03. Problem Statement

Students and small businesses need QR codes constantly — for Wi-Fi sharing, payment collection, labelling — but existing free tools either phone home with the payload data, stamp watermarks on the output, or offer no way to decode a code seen on screen. Meanwhile the payload formats phones expect (the WIFI: URI scheme, vCard 3.0, upi://pay intents) are fiddly to compose by hand, and a malformed string silently produces a code that scans to garbage. The project builds a single offline desktop utility that composes these payloads correctly, renders them with styling control, reads codes back from the screen or image files, and remembers every code it touches.

04. Objectives

• Compose standards-compliant payloads through one-click templates: URL, plain text, Wi-Fi credentials (WPA/WEP/open with proper escaping), vCard 3.0 contacts and UPI payment intents.

• Render codes with automatic QR version selection (1–40), selectable Reed–Solomon error correction (L/M/Q/H), custom ink and paper colours, and an optional embedded centre logo.

• Export as PNG at selectable resolution (256–1024 px) and as true vector SVG generated from the module matrix.

• Decode QR codes from a screen-region capture or PNG/JPEG/WebP image files using the ZBar engine, with payload classification (link, Wi-Fi, contact, UPI, text).

• Generate codes in batch from a pasted list (up to 24 payloads per run) for badges, labels and tents.

• Keep a persistent local history (60 most recent entries) with thumbnails, reload, PNG export and JSON export.

05. Proposed Methodology

The generator pipeline has three stages. First, the payload builder validates the form fields and composes the canonical string: URLs gain a scheme if missing, Wi-Fi fields are escaped per the WIFI: URI rules (; : , and \ are backslash-escaped), contacts are emitted as vCard 3.0, and UPI fields become a upi://pay query string. Second, the qrcode library encodes the string in byte mode, auto-selecting the smallest QR version that fits and applying Reed–Solomon error correction at the user's chosen level. Third, Pillow renders the module matrix at the requested size and colours; when the logo option is on, a white patch covering roughly a fifth of the code width is cleared and the logo composited at the centre, and the UI recommends quartile/high correction to cover the lost modules. The reader pipeline captures a screen region with mss (or opens an image file), converts to grayscale, and hands the frame to pyzbar; on a successful decode the payload is classified by prefix and presented with the matching action. Tkinter hosts both pipelines in a tabbed native GUI, and PyInstaller packages the whole program as a single executable.

06. System Architecture / Working

Three tabs share one window. The Generator tab holds the content-type selector, the dynamic field form, style controls (colours, error correction, export size, logo toggle, batch panel) and a live preview that re-renders on every keystroke, annotated with version, module count and character statistics. The Reader tab offers a file picker and a screen-region capture button; a decoded payload appears with its type badge, the raw text, and Copy / Open-link / Save-to-history actions. The History tab lists every generated and scanned code as a card with thumbnail, type, timestamp and Reload / PNG / Delete actions, plus JSON export and clear-all. A JSON file in the user's config directory persists the history between runs; no network calls exist anywhere in the program.

07. Hardware / Software Requirements

• Any 64-bit Windows 10/11, Linux or macOS machine; Python 3.10+ for development (end users get the packaged executable).

• Python libraries: qrcode, Pillow, pyzbar (needs the ZBar shared library), opencv-python, mss; PyInstaller for packaging.

• Tkinter (ships with standard Python); no other GUI toolkit.

• Screen-capture permission on macOS for the region reader; a webcam is not required.

• No internet connection required at runtime.

08. Expected Outcomes

• A working offline desktop app that generates styled, logo-capable QR codes for five payload types and reads codes back from screen regions and image files.

• Correct-by-construction payloads: escaped Wi-Fi URIs, valid vCard 3.0 and well-formed UPI intents that phone cameras interpret as intended.

• A persistent, searchable history model with reload and JSON export.

• A complete report, PPT and viva Q&A.;

09. Applications

• Sharing Wi-Fi credentials with guests without dictating passwords.

• Generating UPI payment QR codes for small shops and stalls.

• Batch-producing labelled codes for events, inventory and classrooms.

• Teaching QR encoding, Reed–Solomon error correction and image-based decoding in a hands-on desktop project.

10. Future Scope

• Decoding additional 1D barcode formats the ZBar engine already supports.

• Optional webcam-based live scanning alongside screen capture.

• Drag-and-drop of images onto the reader and a system-tray quick-capture hotkey.

• Encrypted history vault for sensitive payloads such as Wi-Fi passwords.