CuraConnect: System Documentation & Implementation Guide (v2.0)

Version 2.0 | October 2024

---

1. Project Overview

CuraConnect is a zero-cost, modern, and serverless patient registration and token management system designed for clinics. It streamlines the patient intake process, reduces administrative workload, and improves the patient experience without any recurring software or hardware costs.

The system allows patients to check in by simply scanning a QR code with their smartphone. Their details are then automatically organized into a live, digital queue in a Google Sheet that the clinic staff can monitor in real-time.

This version (v2.0) represents a significant architectural and security enhancement over the original, introducing a professional CI/CD pipeline and robust security practices.

Core Technologies:

- **Frontend (UI):** GitHub Pages (hosting a static HTML, CSS, & JavaScript file).
- **Backend (API):** Google Apps Script.
- **Database:** Google Sheets.
- **CI/CD & Security:** GitHub Actions.

---

2. System Architecture & How It Works

The system is built on a **decoupled architecture**, which means the user interface (frontend) is completely separate from the data processing logic (backend). This makes the application more stable, secure, and much easier to update.

A. The Frontend (The Patient's View)

- **What it is:** A single index.html file hosted for free on GitHub Pages.
- **How it works:** When a patient scans the QR code, it opens this webpage. The page contains the registration form and the JavaScript logic required to capture the patient's data and send it to the backend via secure fetch API calls. It also handles language switching (English/Marathi) and basic client-side security to deter casual inspection.

B. The Backend (The "Engine")

- **What it is:** A Google Apps Script deployed as a Web App that acts as a secure, private REST API.
- **How it works:** The script's only job is to listen for incoming data from the frontend. When it receives a request, it performs all the critical logic:
    1. Validates the phone number to ensure it is 10 digits.

2. Searches the database to see if the patient is new or returning to assign the correct Patient ID.
3. Calculates the correct token number for the current day.
4. Saves the complete, verified record into the Google Sheet.
5. Handles admin authentication and registration status changes securely on the server.
6. Sends a success or error message back to the frontend in JSON format.

C. The Database (The Staff's View)

- **What it is:** A Google Sheet in your Google Drive.
- **How it works:** This spreadsheet is the single source of truth. It acts as a simple, powerful database where all patient records are stored. For the clinic staff, it serves as a live, real-time dashboard of the patient queue.

D. The CI/CD Pipeline (The Automation)

- **What it is:** A GitHub Actions workflow defined in the .github/workflows/deploy.yml file.
- **How it works:** This is the key to keeping the backend API URL secure.
    1. The actual API URL is stored as an encrypted "Secret" in the GitHub repository settings, not in the code.
    2. When you push new code to the main branch, the workflow automatically runs.
    3. It takes the index.html file, replaces a placeholder with the real API URL from the secret, and then deploys the final, working version to GitHub Pages.

---

3. Implementation Guide: Step-by-Step Setup

Follow these steps to set up the entire system from scratch.

Phase 1: Backend Setup (Google)

1. **Create the Google Sheet:**
    - Go to sheets.new to create a new Google Sheet.
    - Rename the sheet tab at the bottom to be exactly **Patient Data**.
    - In the first row, set up these exact headers in this order: Timestamp, PatientID, Name, Phone, Age, TokenNumber, Date
2. **Create the Apps Script:**
    - In your sheet, go to **Extensions -> Apps Script**.
    - Delete any existing code in the Code.gs file and paste your final, robust backend code (code.gs).

- In the first line of the script, set your desired admin password: const ADMIN_PASSWORD = "your_secure_password";

3. **Deploy the Script:**
    - Click **Deploy -> New deployment**.
    - Set **"Execute as"** to **Me (your email)**.
    - Set **"Who has access"** to **Anyone**.
    - Click **Deploy** and authorize the permissions.
    - Copy the final **"Web app" URL** that ends in /exec. This is your permanent API endpoint. Keep it safe.

Phase 2: Frontend & Deployment Setup (GitHub)

1. **Create GitHub Repository:**
    - Go to GitHub and create a new **public** repository (e.g., CuraConnect-UI).
    - Upload your index.html, .github/workflows/deploy.yml, and logo.png files to this repository.
2. **Create GitHub Secret:**
    - In your new repository, go to **Settings > Secrets and variables > Actions**.
    - Click **New repository secret**.
    - **Name:** SCRIPT_URL
    - **Value:** Paste the /exec URL you copied from the Apps Script deployment.
3. **Enable GitHub Pages:**
    - In the repository settings, go to the **Pages** tab.
    - Under "Build and deployment", change the "Source" to **GitHub Actions**.
4. **Trigger Deployment:**
    - The workflow should have run automatically when you uploaded your files. Go to the **Actions** tab in your repository to check its status.
    - Once the workflow completes successfully, go back to the **Pages** tab in settings. Your live website URL will be displayed at the top.

Phase 3: Finalization

1. **Generate QR Code:** Use a free online QR code generator to create a code from your final GitHub Pages URL.
2. **Print & Display:** Print this QR code and place it at the clinic's reception.

---

4. Precautions & Best Practices

- **Security:** The Google Sheet contains sensitive patient data. **Never** make the sheet itself public. Only share it directly with authorized clinic staff by inviting them via their email address.
- **URL Management:** Your Apps Script /exec URL is your API key. Thanks to the GitHub Actions setup, it is no longer exposed in your source code. Manage it only within GitHub Secrets. The GitHub Pages URL is the only one you need to share publicly.
- **Data Backup:** Once a month, it is good practice to make a copy of your Google Sheet (File -> Make a copy) as a simple backup.
- **Timezone:** Ensure your Google Sheet's timezone (File -> Settings) is set to the same timezone as your script (Asia/Kolkata) to prevent date-related bugs.

---

5. Troubleshooting Common & Critical Issues

This section documents the issues we solved together. If the app breaks, the cause is almost certainly listed here.

Issue	Symptoms	Root Cause	Solution
1. "An error occurred"	The error alert appears on the screen. The browser console shows a 500 Internal Server Error.	The backend Apps Script crashed while running.	**Check the Executions Log in Apps Script.** The most common cause is the sheet tab is not named **Patient Data** or the column headers are in the wrong order.
2. "Failed to fetch"	The error alert appears. The browser console shows a `TypeError: Failed to fetch` or a CORS policy error.	A communication breakdown. The frontend can't talk to the backend.	This is a deployment or URL issue. **Perform a "Triple Check Redeployment"**: 1. Copy the `/exec` URL from Apps Script. 2. Update the `SCRIPT_URL` secret in GitHub. 3. Push a small change to GitHub to trigger a new frontend deployment.
3. New Patient ID Every Time	A returning patient submits the form but gets a new Patient ID instead of their old one.	The script can't match the patient. This is caused by a "Number vs. Text" mismatch with the phone number.	The final, robust `getPatientId` function already solves this by converting both the form's phone number and the sheet's phone number to **strings** before comparing them.
4. Token Number is Always "1"	Every patient for the day gets the token number 1.	The script can't match the dates. This is caused by a date format mismatch between the sheet and the script.	The final, robust `getNextToken` function already solves this by using `Utilities.formatDate` to force both dates into the exact same format (`dd/MM/yyyy`) before comparing.
5. GitHub Action Fails	The workflow in the "Actions" tab fails with a permissions error.	The workflow does not have permission to write to GitHub Pages.	The `permissions` block must be added to the `deploy.yml` file to grant `pages: write` and `id-token: write` access.