# QR Scanner System — Technical Documentation

## 🧩 1. System Architecture Overview

### Components

- **React Frontend**
- **FastAPI Backend**
- **PostgreSQL Database**
- **(Optional) Redis Cache**
- **Staff Devices** → Mobile Browser (Camera enabled)

### Flow

1. Staff logs into React App → receives JWT.
2. Selects voucher type (Entry / Food / Gift / Store).
3. Camera opens → QR is scanned.
4. React sends QR + voucher type + staff role → Backend.
5. Backend validates → records `scan_log` → returns result.
6. UI shows Success / Fail.

---

## 📦 2. Frontend (React) Technical Documentation

### 2.1 Tech Stack

- React (Vite preferred)
- TailwindCSS
- Axios
- html5-qrcode (for scanning)
- Redux Toolkit (optional)
- JWT authentication

### 2.2 Folder Structure

```
src/
  api/
    auth.js
    scan.js
  components/
    QRScanner.jsx
    ResultCard.jsx
  pages/
    Login.jsx
    VoucherSelect.jsx
    Scanner.jsx
  utils/
    auth.js
```

```
  styles/
    global.css
```

### 2.3 QR Scanner Component

#### Scanner Workflow

1. Start camera using html5-qrcode
2. On QR detection:
    - Freeze scan
    - Send API request
    - Redirect to result page

#### Sample Code Snippet

```
import { Html5QrcodeScanner } from "html5-qrcode";

function QRScanner({ voucherType }) {
  useEffect(() => {
    const scanner = new Html5QrcodeScanner(
      "reader",
      { fps: 25, qrbox: 250 },
      false
    );

    scanner.render(async (qrData) => {
      await sendScanRequest(qrData, voucherType);
    });

  }, []);

  return <div id="reader" className="w-full"></div>;
}
```

---

## 🧠 3. Backend (FastAPI) Technical Documentation

### 3.1 Tech Stack

- FastAPI
- PostgreSQL (asyncpg or psycopg)
- SQLAlchemy ORM
- Pydantic
- JWT (python-jose)
- Uvicorn

### 3.2 Folder Structure

```
app/
  main.py
  auth/
    routes.py
    service.py
  scan/
    routes.py
    service.py
  models/
    staff.py
    tickets.py
    vouchers.py
    scan_logs.py
  db/
    config.py
  utils/
    jwt.py
    qr_decoder.py
```

---

## 🧾 4. Database Schema (PostgreSQL)

### 4.1 staff

| Column | Type | Description |
|---|---|---|
| id | bigserial PK | Staff ID |
| name | text | Staff name |
| email | text unique | Login email |
| password | text | Hashed |
| role | text | gate / food / gift / store |
| status | smallint | 1 = active |

### 4.2 tickets

**Purpose:** Store unknown tickets scanned at gate.

| Column | Type | Description |
|---|---|---|
| id | bigserial PK | Internal ticket ID |
| ticket_id | text | From QR |
| event_id | text | From QR |
| created_at | timestamp | Auto |

### 4.3 ticket_vouchers

Each ticket has 4 rows, one per voucher type.

| Column | Type | Description |
|---|---|---|
| id | bigserial | PK |
| ticket_id | bigint FK | From tickets table |
| voucher_type | text | entry / food / gift / store |
| is_used | bool | True after redeem |
| used_at | timestamp | Redeemed time |

### 4.4 scan_logs

Logs every scan attempt.

| Column | Type | Description |
|---|---|---|
| id | bigserial | PK |
| ticket_id | bigint | Ticket scanned |
| staff_id | bigint | Who scanned |
| voucher_type | text | entry / food / gift / store |
| status | text | success / failed |
| reason | text | 'already_used', 'invalid', etc. |
| scanned_at | timestamp | Auto |

---

## 🔐 5. Backend API Design

### 5.1 Auth API

**POST** `/auth/login`

**Request:**

```
{
  "email": "staff@mail.com",
  "password": "123456"
}
```

**Response:**

```
{
  "token": "JWT_TOKEN",
```

```
  "role": "food"
}
```

### 5.2 Ticket Scan API

**POST** `/scan/validate`

**Request:**

```
{
  "qr_raw": "QR_CONTENT_HERE",
  "voucher_type": "food"
}
```

**Backend Logic**

1. Decode QR
2. Extract `ticket_id`, `event_id`
3. If ticket not exists → create ticket
4. Check if staff role == voucher_type
5. Check `ticket_vouchers.is_used`
6. If not used → mark used
7. Insert `scan_log`
8. Return result

**Response (Success)**

```
{
  "status": "success",
  "message": "Voucher redeemed",
  "ticket_id": "TCK1001",
  "voucher_type": "food",
  "timestamp": "2026-02-21T12:09:55"
}
```

**Response (Fail)**

```
{
  "status": "failed",
  "reason": "already_used"
}
```

---

## ⚙ 6. QR Parsing Logic

Since QR format is not finalized, we support all 3 types:

| Type | Description |
|---|---|
| **Type A — JSON QR** | `{ "ticket_id": "123", "event_id": "E5", "food": 1 }` |
| **Type B — Encrypted QR** | Hex/Base64 → Decrypted via shared key |
| **Type C — Ticket-ID only** | Backend creates voucher rows on first scan. |

---

## 👮 7. Roles & Permissions

| Role | Allowed Voucher |
|---|---|
| gate_staff | entry |
| food_staff | food |
| gift_staff | gift |
| store_staff | store |

**Backend enforces:** If staff tries scanning other vouchers → **FAIL**

---

## 📊 8. Admin Panel (Phase-2)

### Features

- Event summary
- Total scans per type
- Failed scans
- Staff analytics
- Export CSV
- Real-time dashboard

---

## 🚀 9. Performance & Scaling

With 5k–20k tickets:

- PostgreSQL with indexing is enough
- Redis caching for repeated QR scans (optional)
- API response time: **< 80ms**
- Scanning speed: **25–30 frames/sec**

---

## ☁️ 10. Deployment Plan

| Layer | Option |
|---|---|
| Frontend | Vercel / Netlify / AWS S3 hosting |

| Layer | Option |
|---|---|
| Backend | AWS EC2 Ubuntu 22.04, Uvicorn + Nginx, HTTPS with Certbot |
| Database | AWS RDS PostgreSQL OR Supabase (cheaper) |
| Domain | Cloudflare DNS |

**Scanner URL:** `https://scanner.yourevent.com`