QRSecure - Complete Project Documentation

Project Overview

Project Name: QRSecure
Type: QR Code Management Platform with Analytics, Dynamic URLs, and Security Features
Tech Stack: Python (Backend), React/HTML-CSS-JS (Frontend), PostgreSQL (Database)
Estimated Development Time: 2-3 weeks for MVP

Core Problem Being Solved

Traditional QR code generators create static codes with:

- No analytics tracking
- No ability to update destination URLs
- No security validation
- No preview mechanism for users

QRSecure solves all of these problems by using an intermediary redirect system that enables analytics, dynamic URL updates, and security checking.

Key Features

Feature 1: Analytics Tracking

- Track every scan with detailed metrics
- Capture: timestamp, location, device type, browser, OS
- Display analytics dashboard with charts and graphs
- Real-time scan counting

Feature 2: Dynamic QR Codes

- Update destination URLs without regenerating QR codes
- QR codes remain functional even when destination changes
- Version history of URL changes

- Instant updates (no delays)

Feature 3: Security & Preview

- Preview page showing destination before redirect
- Google Safe Browsing API integration
- SSL certificate validation
- Domain age checking
- Suspicious URL detection
- User reporting mechanism

Feature 4: Customization (Bonus)

- Custom QR code colors
- Logo embedding in QR codes
- Different QR code styles
- Download in multiple formats (PNG, SVG)

Architecture Overview

System Flow

```
User Input (URL) → Backend Validation → Database Storage → QR Generation → User Downloads QR
                                                                                    ↓
Someone Scans QR → Phone Camera → Your Server → Analytics Logging → Preview Page → Final Redirect → Destination
```

Why This Architecture?

The QR code doesn't point directly to the destination. Instead:

- QR Code contains: `https://qrsecure.app/xK9mP` (your short URL)
- Your server intercepts the request
- Logs analytics data
- Performs security checks

- Shows preview page (optional)
- Redirects to actual destination

Complete Technology Stack

Backend

- **Framework:** FastAPI (Python)
  - Fast, modern, async support
  - Automatic API documentation
  - Easy to learn
- **Language:** Python 3.10+
- **Server:** Uvicorn (ASGI server)

Database

- **Database:** PostgreSQL
- **Hosting:** Supabase (free tier)
- **ORM:** SQLAlchemy (Python library for database operations)

Frontend

- **Option 1:** React (more professional, harder)
- **Option 2:** HTML + CSS + JavaScript (simpler, faster to build)
- **Styling:** Tailwind CSS
- **Charts:** Chart.js (for analytics visualization)

QR Code Generation

- **Library:** `qrcode` (Python)
- **Image Processing:** `Pillow` (PIL)

Authentication

- **Solution:** Supabase Auth (built-in authentication)

- **Features:** Email/password, OAuth (Google, GitHub)
- **No need to build authentication from scratch**

Analytics & Security

- **Geolocation:** GeoLite2 (free MaxMind database)
- **User Agent Parsing:** `user-agents` library
- **Security:** Google Safe Browsing API
- **SSL Checking:** Python `ssl` and `socket` libraries
- **WHOIS Lookup:** `python-whois` library

Hosting

- **Backend:** Railway.app or Render.com (free tier)
- **Frontend:** Vercel or Netlify (free tier)
- **Database:** Supabase (free tier)

Database Schema

Table 1: users

```sql
CREATE TABLE users (
    user_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    subscription_tier VARCHAR(50) DEFAULT 'free'
);
```

Table 2: urls

```sql
```

```sql
CREATE TABLE urls (
    url_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(user_id),
    short_code VARCHAR(10) UNIQUE NOT NULL,
    original_url TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    is_active BOOLEAN DEFAULT true,
    show_preview BOOLEAN DEFAULT true,
    analytics_enabled BOOLEAN DEFAULT true,
    custom_title VARCHAR(255),
    password_hash VARCHAR(255),
    expiration_date TIMESTAMP,
    total_scans INTEGER DEFAULT 0
);
```

Table 3: scans

```sql
CREATE TABLE scans (
    scan_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    url_id UUID REFERENCES urls(url_id),
    scanned_at TIMESTAMP DEFAULT NOW(),
    ip_address VARCHAR(45),
    country VARCHAR(100),
    city VARCHAR(100),
    device_type VARCHAR(50),
    os VARCHAR(50),
    browser VARCHAR(50),
    user_agent TEXT
);
```

Table 4: url_history

```sql
```

```
CREATE TABLE url_history (
    history_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    url_id UUID REFERENCES urls(url_id),
    old_url TEXT,
    new_url TEXT,
    changed_at TIMESTAMP DEFAULT NOW()
);
```

Complete User Workflows

Workflow 1: Creating a QR Code

Step 1: User visits `https://qrsecure.app`

Step 2: User signs up or logs in (Supabase handles this)

Step 3: User sees QR creation form:

- Input field: "Enter destination URL"
- Checkbox: "Show preview page before redirect" (checked by default)
- Checkbox: "Enable analytics" (checked by default)
- Optional: Custom title, expiration date, password protection
- Button: "Generate QR Code"

Step 4: User enters URL: `https://nike.com/winter-sale`

Step 5: User clicks "Generate QR Code"

Backend Process:

```
python
```

```
# 1. Validate URL format
if not is_valid_url(url):
    return error("Invalid URL format")

# 2. Check if URL is reachable
try:
    response = requests.head(url, timeout=5)
    if response.status_code >= 400:
        return error("URL not reachable")
except:
    return error("Cannot connect to URL")

# 3. Security check with Google Safe Browsing
threat_result = check_safe_browsing(url)
if threat_result.is_malicious:
    return error("URL flagged as malicious")

# 4. Check SSL certificate
ssl_valid = check_ssl_certificate(url)

# 5. Generate unique short code
short_code = generate_short_code()  # e.g., "xK9mP"

# 6. Store in database
db.insert(urls, {
    user_id: current_user.id,
    short_code: short_code,
    original_url: url,
    show_preview: true,
    analytics_enabled: true,
    is_active: true
})

# 7. Generate QR code image
qr = qrcode.QRCode(version=1, box_size=10, border=4)
qr.add_data(f"https://qrsecure.app/{short_code}")
qr.make(fit=True)
img = qr.make_image(fill_color="black", back_color="white")
img.save(f"qr_codes/{short_code}.png")

# 8. Return QR code and details to user
return {
    short_url: f"https://qrsecure.app/{short_code}",
```

```
    qr_image: f"/qr_codes/{short_code}.png",
    destination: url,
    created_at: now()
}
```

Step 6: User sees dashboard with:

- QR code image (can download as PNG or SVG)
- Short URL: `qrsecure.app/xK9mP`
- Destination: `nike.com/winter-sale`
- Scan count: 0
- Buttons: Download, Edit, Delete, View Analytics

Workflow 2: Scanning a QR Code

Step 1: Person scans QR code with phone camera

Step 2: Phone detects URL: `https://qrsecure.app/xK9mP`

Step 3: Browser sends GET request to your server

Backend Process:

```
python
```

```
@app.get("/{short_code}")
async def redirect_url(short_code: str, request: Request):
    # 1. Look up short code in database
    url_data = db.query(urls).filter(short_code == short_code).first()

    # 2. Check if code exists and is active
    if not url_data or not url_data.is_active:
        return "QR code not found or inactive"

    # 3. Check expiration
    if url_data.expiration_date and url_data.expiration_date < now():
        return "QR code has expired"

    # 4. Extract analytics data
    ip = request.client.host
    user_agent = request.headers.get("user-agent")

    # 5. Parse device information
    device_info = parse_user_agent(user_agent)

    # 6. Get location from IP
    location = get_location(ip)

    # 7. Save scan data to database
    db.insert(scans, {
        url_id: url_data.url_id,
        ip_address: ip,
        country: location.country,
        city: location.city,
        device_type: device_info.device,
        os: device_info.os,
        browser: device_info.browser,
        user_agent: user_agent
    })

    # 8. Increment scan counter
    db.update(urls).set(total_scans += 1).where(url_id == url_data.url_id)

    # 9. Check if preview page is enabled
    if url_data.show_preview:
        return render_preview_page(url_data)
```

```
    else:
        return RedirectResponse(url=url_data.original_url, status_code=302)
```

Step 4: If preview page is enabled, user sees:

```
html
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Link Preview - QRSecure</title>
</head>
<body>
  <div class="preview-container">
    <h2>🔒 Link Preview</h2>
    <p>You're about to visit:</p>
    <h3>nike.com/winter-sale</h3>

    <div class="security-badges">
      ✓ Secure (HTTPS)
      ✓ Verified Safe
      ✓ Domain Age: 28 years
    </div>

    <div class="metadata">
      Created by: Nike Official
      Last updated: Jan 3, 2026
    </div>

    <button onclick="continueToSite()">Continue to Site</button>
    <button onclick="goBack()">Go Back</button>

    <a href="/report?code=xK9mP">🚩 Report suspicious link</a>
  </div>

  <script>
    // Auto-redirect after 3 seconds (optional)
    setTimeout(() => {
      continueToSite();
    }, 3000);

    function continueToSite() {
      window.location.href = "https://nike.com/winter-sale";
    }

    function goBack() {
      window.history.back();
    }
  </script>
```

```
</body>
</html>
```

Step 5: User clicks "Continue to Site" or waits 3 seconds

Step 6: Redirect to `https://nike.com/winter-sale` ✅

Workflow 3: Viewing Analytics

Step 1: User logs into QRSecure dashboard

Step 2: User clicks "View Analytics" for a specific QR code

Step 3: Backend queries scan data:

```python
```

```
@app.get("/api/analytics/{short_code}")
async def get_analytics(short_code: str):
    # Get URL data
    url = db.query(urls).filter(short_code == short_code).first()

    # Get all scans
    scans = db.query(scans).filter(url_id == url.url_id).all()

    # Calculate metrics
    total_scans = len(scans)
    scans_today = count_scans_today(scans)

    # Group by location
    locations = group_by(scans, 'country')

    # Group by device
    devices = group_by(scans, 'device_type')

    # Scans over time (last 30 days)
    timeline = get_scans_timeline(scans, days=30)

    return {
        total_scans: total_scans,
        scans_today: scans_today,
        locations: locations,
        devices: devices,
        timeline: timeline
    }
```

Step 4: Frontend displays:

- Total scans counter
- Today's scans
- Line chart: scans over time
- Pie chart: device distribution
- Map or list: top countries/cities
- Table: recent scans with details

Workflow 4: Updating Destination URL (Dynamic Feature)

Step 1: User goes to dashboard

Step 2: User clicks "Edit" button for a QR code

Step 3: User sees modal:

```
Current destination: https://nike.com/winter-sale
New destination: [https://nike.com/spring-collection]

[Save Changes]  [Cancel]
```

Step 4: User enters new URL and clicks "Save Changes"

Backend Process:

```python
```

```
@app.put("/api/urls/{short_code}")
async def update_url(short_code: str, new_url: str):
    # 1. Validate new URL
    if not is_valid_url(new_url):
        return error("Invalid URL")

    # 2. Security check
    if not check_safe_browsing(new_url).is_safe:
        return error("URL flagged as malicious")

    # 3. Get current URL data
    url_data = db.query(urls).filter(short_code == short_code).first()

    # 4. Save to history
    db.insert(url_history, {
        url_id: url_data.url_id,
        old_url: url_data.original_url,
        new_url: new_url
    })

    # 5. Update URL
    db.update(urls).set(
        original_url = new_url,
        updated_at = now()
    ).where(short_code == short_code)

    return success("URL updated successfully")
```

Step 5: Confirmation message: "Destination updated! All scans now redirect to new URL."

Important: The QR code image never changes. Anyone scanning the same QR code will now be redirected to the new destination.

Security Implementation Details

Google Safe Browsing API

Setup:

1. Go to: https://cloud.google.com/security/products/safe-browsing
2. Create Google Cloud account (free)

3. Enable Safe Browsing API
4. Get API key

Free Tier:

- 10,000 requests per day
- No cost

Implementation:

```python
import requests

def check_safe_browsing(url):
    api_key = "YOUR_API_KEY"
    endpoint = "https://safebrowsing.googleapis.com/v4/threatMatches:find"

    payload = {
        "client": {
            "clientId": "qrsecure",
            "clientVersion": "1.0"
        },
        "threatInfo": {
            "threatTypes": ["MALWARE", "SOCIAL_ENGINEERING", "UNWANTED_SOFTWARE"],
            "platformTypes": ["ANY_PLATFORM"],
            "threatEntryTypes": ["URL"],
            "threatEntries": [{"url": url}]
        }
    }

    response = requests.post(f"{endpoint}?key={api_key}", json=payload)
    data = response.json()

    if "matches" in data:
        return {"is_safe": False, "threats": data["matches"]}
    return {"is_safe": True}
```

SSL Certificate Check

```python
```

```python
import ssl
import socket
from urllib.parse import urlparse

def check_ssl_certificate(url):
    domain = urlparse(url).netloc

    try:
        context = ssl.create_default_context()
        with socket.create_connection((domain, 443), timeout=5) as sock:
            with context.wrap_socket(sock, server_hostname=domain) as ssock:
                cert = ssock.getpeercert()

                # Check expiration
                not_after = cert['notAfter']
                # Parse date and check if expired

                return {
                    "has_ssl": True,
                    "issuer": cert.get('issuer'),
                    "valid_until": not_after
                }
    except:
        return {"has_ssl": False}
```

Domain Age Check

```python
```

```python
import whois
from datetime import datetime

def check_domain_age(url):
    domain = urlparse(url).netloc

    try:
        w = whois.whois(domain)
        creation_date = w.creation_date

        if isinstance(creation_date, list):
            creation_date = creation_date[0]

        age_days = (datetime.now() - creation_date).days

        # Flag if domain is less than 30 days old
        is_suspicious = age_days < 30

        return {
            "age_days": age_days,
            "created": creation_date,
            "is_new": is_suspicious
        }
    except:
        return {"age_days": None, "error": "Could not fetch domain info"}
```

Authentication System (Supabase)

Why Supabase?

- Free authentication built-in
- No need to code login/signup from scratch
- Handles password hashing, email verification
- Supports OAuth (Google, GitHub login)
- Includes database (PostgreSQL)

Setup Process:

Step 1: Go to https://supabase.com

Step 2: Create free account

Step 3: Create new project

- Project name: qrsecure
- Database password: (set strong password)
- Region: (closest to you)

Step 4: Get credentials:

- Project URL: `https://xyzproject.supabase.co`
- API Key (anon public): `eyJhbGc...`
- Service role key: `eyJhbGc...` (keep secret)

Step 5: Install Supabase client:

```bash
pip install supabase
```

Step 6: Initialize in Python:

```python
from supabase import create_client, Client

url = "https://xyzproject.supabase.co"
key = "your-anon-key"
supabase: Client = create_client(url, key)
```

Authentication Functions:

Sign Up:

```python
```

```python
def sign_up(email, password):
    try:
        user = supabase.auth.sign_up({
            "email": email,
            "password": password
        })
        return user
    except Exception as e:
        return {"error": str(e)}
```

Login:

```python
def login(email, password):
    try:
        session = supabase.auth.sign_in_with_password({
            "email": email,
            "password": password
        })
        return session
    except Exception as e:
        return {"error": str(e)}
```

Get Current User:

```python
def get_current_user():
    user = supabase.auth.get_user()
    return user
```

Logout:

```python
def logout():
    supabase.auth.sign_out()
```

Frontend Integration (JavaScript):

```javascript
```

```
// Initialize Supabase client
import { createClient } from '@supabase/supabase-js'

const supabase = createClient(
  'https://xyzproject.supabase.co',
  'your-anon-key'
)

// Sign up
async function signUp(email, password) {
  const { data, error } = await supabase.auth.signUp({
    email: email,
    password: password
  })
  if (error) console.error(error)
  return data
}

// Login
async function login(email, password) {
  const { data, error } = await supabase.auth.signInWithPassword({
    email: email,
    password: password
  })
  if (error) console.error(error)
  return data
}

// Check if user is logged in
async function checkUser() {
  const { data: { user } } = await supabase.auth.getUser()
  return user
}
```

Geolocation Setup (GeoLite2)

Download Database:

Step 1: Go to https://dev.maxmind.com/geoip/geolite2-free-geolocation-data

Step 2: Create free account

Step 3: Download GeoLite2 City database (MMDB format)

Step 4: Install Python library:

```bash
pip install geoip2
```

Step 5: Implementation:

```python
import geoip2.database

# Load database
reader = geoip2.database.Reader('GeoLite2-City.mmdb')

def get_location(ip_address):
    try:
        response = reader.city(ip_address)

        return {
            "country": response.country.name,
            "country_code": response.country.iso_code,
            "city": response.city.name,
            "latitude": response.location.latitude,
            "longitude": response.location.longitude
        }
    except:
        return {
            "country": "Unknown",
            "city": "Unknown"
        }

# Example usage
location = get_location("8.8.8.8")
print(location)  # {'country': 'United States', 'city': 'Mountain View', ...}
```

Free Tier:

- Completely free
- Update database monthly (it's static data)
- Unlimited lookups

Complete Installation & Setup Guide

Step 1: Install Python Dependencies

Create `requirements.txt`:

```
fastapi==0.104.1
uvicorn[standard]==0.24.0
sqlalchemy==2.0.23
psycopg2-binary==2.9.9
python-dotenv==1.0.0
qrcode[pil]==7.4.2
Pillow==10.1.0
requests==2.31.0
geoip2==4.7.0
user-agents==2.2.0
python-whois==0.8.0
supabase==2.0.3
pydantic==2.5.0
```

Install:

```bash
pip install -r requirements.txt
```

Step 2: Environment Variables

Create `.env` file:

```
# Supabase
SUPABASE_URL=https://your-project.supabase.co
SUPABASE_KEY=your-anon-key
SUPABASE_SERVICE_KEY=your-service-key

# Google Safe Browsing
GOOGLE_SAFE_BROWSING_KEY=your-api-key

# Database (Supabase PostgreSQL)
DATABASE_URL=postgresql://postgres:password@db.your-project.supabase.co:5432/postgres

# App Settings
APP_URL=https://qrsecure.app
SECRET_KEY=your-random-secret-key
```

Step 3: Project Structure

```
qrsecure/
├── backend/
│   ├── main.py              # FastAPI app entry point
│   ├── models.py            # Database models
│   ├── routes/
│   │   ├── auth.py          # Authentication routes
│   │   ├── qr.py            # QR code generation routes
│   │   ├── analytics.py     # Analytics routes
│   │   └── redirect.py      # Redirect handling
│   ├── services/
│   │   ├── qr_generator.py  # QR code generation logic
│   │   ├── security.py      # Security checks
│   │   ├── analytics.py     # Analytics processing
│   │   └── geolocation.py   # IP to location
│   ├── database.py          # Database connection
│   └── config.py            # Configuration
├── frontend/
│   ├── index.html           # Landing page
│   ├── dashboard.html       # User dashboard
│   ├── css/
│   │   └── styles.css
│   └── js/
│       ├── auth.js          # Authentication
│       ├── qr-creator.js    # QR code creation
│       └── analytics.js     # Analytics display
```

```
├── qr_codes/                # Generated QR images
├── GeoLite2-City.mmdb       # Geolocation database
├── requirements.txt
├── .env
└── README.md
```

Step 4: Database Setup (Supabase)

In Supabase SQL Editor, run:

sql

```sql
-- Create users table (Supabase auth handles this, but add custom fields)
CREATE TABLE user_profiles (
    user_id UUID PRIMARY KEY REFERENCES auth.users(id),
    subscription_tier VARCHAR(50) DEFAULT 'free',
    created_at TIMESTAMP DEFAULT NOW()
);

-- Create urls table
CREATE TABLE urls (
    url_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES auth.users(id),
    short_code VARCHAR(10) UNIQUE NOT NULL,
    original_url TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    is_active BOOLEAN DEFAULT true,
    show_preview BOOLEAN DEFAULT true,
    analytics_enabled BOOLEAN DEFAULT true,
    custom_title VARCHAR(255),
    password_hash VARCHAR(255),
    expiration_date TIMESTAMP,
    total_scans INTEGER DEFAULT 0
);

-- Create scans table
CREATE TABLE scans (
    scan_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    url_id UUID REFERENCES urls(url_id) ON DELETE CASCADE,
    scanned_at TIMESTAMP DEFAULT NOW(),
    ip_address VARCHAR(45),
    country VARCHAR(100),
    city VARCHAR(100),
    device_type VARCHAR(50),
    os VARCHAR(50),
    browser VARCHAR(50),
    user_agent TEXT
);

-- Create url_history table
CREATE TABLE url_history (
    history_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    url_id UUID REFERENCES urls(url_id) ON DELETE CASCADE,
    old_url TEXT,
```

```
    new_url TEXT,
    changed_at TIMESTAMP DEFAULT NOW()
);

-- Create indexes for performance
CREATE INDEX idx_urls_short_code ON urls(short_code);
CREATE INDEX idx_urls_user_id ON urls(user_id);
CREATE INDEX idx_scans_url_id ON scans(url_id);
CREATE INDEX idx_scans_scanned_at ON scans(scanned_at);
```

Step 5: Run Development Server

```
bash

cd backend
uvicorn main:app --reload --host 0.0.0.0 --port 8000
```

Access at: `http://localhost:8000`

Free Tier Limits & Costs

Supabase (Database + Auth)

- **Free Tier:**
  - 500MB database storage
  - 50,000 monthly active users
  - 2GB bandwidth
  - Unlimited API requests
- **Cost to upgrade:** $25/month for Pro (if you exceed limits)
- **Where to get:** https://supabase.com

Google Safe Browsing API

- **Free Tier:**
  - 10,000 requests per day
  - That's 300,000 per month
- **Cost to upgrade:** No paid tier, hard limit

- **Solution:** Cache results for 24 hours to reduce API calls
- **Where to get:** https://console.cloud.google.com

GeoLite2 Database

- **Free Tier:**
  - Completely free
  - Unlimited lookups
  - Update monthly
- **Cost to upgrade:** MaxMind has paid version (GeoIP2) with more accuracy
- **Where to get:** https://dev.maxmind.com

Railway.app (Backend Hosting)

- **Free Tier:**
  - $5 free credit per month
  - 512MB RAM
  - Shared CPU
  - Good for ~10,000 requests/month
- **Cost to upgrade:** $5/month for more resources
- **Alternative:** Render.com (750 hours free per month)
- **Where to get:** https://railway.app

Vercel (Frontend Hosting)

- **Free Tier:**
  - 100GB bandwidth per month
  - Unlimited websites
  - Automatic HTTPS
  - Good for millions of page views
- **Cost to upgrade:** $20/month for Pro (if you need more)
- **Where to get:** https://vercel.com

Domain Name

- **Cost:** ~$10-15 per year
- **Where to get:** Namecheap, Google Domains, Cloudflare
- **Free alternative:** Use Railway/Vercel subdomain (e.g., qrsecure.up.railway.app)

Total Monthly Cost (Free Tier):

$0 if you stay within limits

Breaking point: When you exceed:

- 300,000 QR code security checks per month (Safe Browsing limit)
- 50,000 active users (Supabase limit)
- 500MB database (Supabase limit)

At scale (10,000 scans/day):

- Supabase: Still free
- Safe Browsing: Still free (with caching)
- Hosting: $5-10/month
- Domain: $10/year
- **Total: ~$5-10/month**

Development Phases

Phase 1: Core Redirect System (2-3 days)

- Set up FastAPI backend
- Create database schema in Supabase
- Implement short URL generation
- Build redirect endpoint
- Test with manual URL entry

Deliverable: Working redirect system (URL → short code → redirect)

Phase 2: QR Code Generation (1-2 days)

- Integrate `qrcode` library
- Create QR generation endpoint
- Save QR images
- Build simple HTML form for URL input
- Display generated QR code

Deliverable: Users can generate QR codes

Phase 3: Authentication (2-3 days)

- Set up Supabase Auth
- Create signup/login pages
- Implement session management
- Connect QR codes to user accounts
- Build user dashboard

Deliverable: Users can sign up and manage their QR codes

Phase 4: Analytics (3-4 days)

- Capture scan data (IP, user agent, timestamp)
- Implement geolocation lookup
- Parse user agent for device info
- Build analytics dashboard with charts
- Create API endpoints for analytics data

Deliverable: Full analytics tracking and visualization

Phase 5: Security Features (3-4 days)

- Integrate Google Safe Browsing API
- Build SSL certificate checker
- Implement domain age verification
- Create preview page template

- Add security badges to preview

Deliverable: Security validation and preview page

Phase 6: Polish & Features (3-5 days)

- Add dynamic URL updating
- Implement QR code customization (colors, logo)
- Build URL history tracking
- Add expiration dates
- Create download options (PNG, SVG)
- Responsive design for mobile

Deliverable: Production-ready MVP

API Endpoints Reference

Authentication

```
POST   /api/auth/signup       - Create new account
POST   /api/auth/login        - User login
POST   /api/auth/logout       - User logout
GET    /api/auth/me           - Get current user
```

QR Code Management

```
POST   /api/qr/create         - Create new QR code
GET    /api/qr/list           - Get all user's QR codes
GET    /api/qr/{short_code}   - Get specific QR code details
PUT    /api/qr/{short_code}   - Update destination URL
DELETE /api/qr/{short_code}   - Delete QR code
GET    /api/qr/{short_code}/download - Download QR image
```

Analytics

```
GET    /api/analytics/{short_code}          - Get analytics for QR code
GET    /api/analytics/{short_code}/timeline  - Get scans over time
```

```
GET    /api/analytics/{short_code}/locations - Get geographic data
GET    /api/analytics/{short_code}/devices   - Get device breakdown
```

Redirect (Public)

```
GET    /{short_code}          - Redirect to destination (main endpoint)
GET    /preview/{short_code}  - Show preview page
POST   /report/{short_code}   - Report malicious link
```

Example Code Snippets

Backend: main.py (FastAPI Entry Point)

```python
```

```python
from fastapi import FastAPI, Request, HTTPException
from fastapi.responses import RedirectResponse, HTMLResponse
from fastapi.staticfiles import StaticFiles
from fastapi.middleware.cors import CORSMiddleware
from dotenv import load_dotenv
import os

# Load environment variables
load_dotenv()

# Initialize FastAPI app
app = FastAPI(title="QRSecure API", version="1.0.0")

# CORS middleware (allow frontend to communicate)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # In production, specify your frontend domain
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Mount static files (QR code images)
app.mount("/qr_codes", StaticFiles(directory="qr_codes"), name="qr_codes")

# Import routes
from routes import auth, qr, analytics, redirect

# Include routers
app.include_router(auth.router, prefix="/api/auth", tags=["Authentication"])
app.include_router(qr.router, prefix="/api/qr", tags=["QR Codes"])
app.include_router(analytics.router, prefix="/api/analytics", tags=["Analytics"])
app.include_router(redirect.router, tags=["Redirect"])

# Root endpoint
@app.get("/")
async def root():
    return {"message": "QRSecure API is running", "version": "1.0.0"}

# Health check
@app.get("/health")
async def health():
    return {"status": "healthy"}
```

```python
if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Backend: routes/redirect.py (Main Redirect Logic)

python

```python
from fastapi import APIRouter, Request, HTTPException
from fastapi.responses import RedirectResponse, HTMLResponse
from services.analytics import log_scan
from services.security import get_security_info
from database import get_db
from models import URLs
import os

router = APIRouter()

@router.get("/{short_code}")
async def redirect_url(short_code: str, request: Request):
    """
    Main redirect endpoint - handles QR code scans
    """
    db = get_db()

    # 1. Look up short code in database
    url_data = db.query(URLs).filter(URLs.short_code == short_code).first()

    if not url_data:
        raise HTTPException(status_code=404, detail="QR code not found")

    if not url_data.is_active:
        raise HTTPException(status_code=410, detail="QR code has been deactivated")

    # 2. Check expiration
    if url_data.expiration_date:
        from datetime import datetime
        if datetime.now() > url_data.expiration_date:
            raise HTTPException(status_code=410, detail="QR code has expired")

    # 3. Log analytics (async, non-blocking)
    if url_data.analytics_enabled:
        await log_scan(
            url_id=url_data.url_id,
            ip_address=request.client.host,
            user_agent=request.headers.get("user-agent", ""),
            db=db
        )

    # 4. Check if preview page is enabled
    if url_data.show_preview:
```

```
        # Redirect to preview page
        return RedirectResponse(url=f"/preview/{short_code}")
    else:
        # Direct redirect
        return RedirectResponse(url=url_data.original_url, status_code=302)


@router.get("/preview/{short_code}")
async def preview_page(short_code: str):
    """
    Show preview page before redirecting
    """
    db = get_db()

    url_data = db.query(URLs).filter(URLs.short_code == short_code).first()

    if not url_data:
        raise HTTPException(status_code=404, detail="QR code not found")

    # Get security information
    security_info = get_security_info(url_data.original_url)

    # Render HTML preview page
    html_content = f"""
    <!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Link Preview - QRSecure</title>
        <style>
            body {{
                font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif;
                display: flex;
                justify-content: center;
                align-items: center;
                min-height: 100vh;
                margin: 0;
                background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            }}
            .preview-container {{
                background: white;
                padding: 40px;
                border-radius: 20px;
```

```
    box-shadow: 0 20px 60px rgba(0,0,0,0.3);
    max-width: 500px;
    text-align: center;
}}
.lock-icon {{
    font-size: 48px;
    margin-bottom: 20px;
}}
h2 {{
    color: #333;
    margin-bottom: 10px;
}}
.destination {{
    font-size: 20px;
    color: #667eea;
    font-weight: bold;
    margin: 20px 0;
    word-break: break-all;
}}
.security-badges {{
    display: flex;
    flex-direction: column;
    gap: 10px;
    margin: 20px 0;
    text-align: left;
}}
.badge {{
    display: flex;
    align-items: center;
    gap: 10px;
    padding: 10px;
    background: #f0f4ff;
    border-radius: 8px;
}}
.badge.success {{ background: #d4edda; color: #155724; }}
.badge.warning {{ background: #fff3cd; color: #856404; }}
.continue-btn {{
    background: #667eea;
    color: white;
    border: none;
    padding: 15px 40px;
    font-size: 16px;
    border-radius: 10px;
    cursor: pointer;
```

```
            margin: 20px 10px 10px;
            transition: background 0.3s;
        }}
        .continue-btn:hover {{
            background: #5568d3;
        }}
        .back-btn {{
            background: #e0e0e0;
            color: #333;
            border: none;
            padding: 15px 40px;
            font-size: 16px;
            border-radius: 10px;
            cursor: pointer;
            transition: background 0.3s;
        }}
        .back-btn:hover {{
            background: #d0d0d0;
        }}
        .report-link {{
            display: block;
            margin-top: 20px;
            color: #666;
            text-decoration: none;
            font-size: 14px;
        }}
        .countdown {{
            color: #888;
            font-size: 14px;
            margin-top: 10px;
        }}
    </style>
</head>
<body>
    <div class="preview-container">
        <div class="lock-icon">🔒</div>
        <h2>Link Preview</h2>
        <p>You're about to visit:</p>
        <div class="destination">{url_data.original_url}</div>

        <div class="security-badges">
            <div class="badge {'success' if security_info['has_ssl'] else 'warning'}">
                <span>{'✓' if security_info['has_ssl'] else '⚠'}</span>
                <span>{'Secure (HTTPS)' if security_info['has_ssl'] else 'Not Secure (HTTP)'}</span>
```

```
            </div>
            <div class="badge {'success' if security_info['is_safe'] else 'warning'}">
                <span>{'✓' if security_info['is_safe'] else '⚠'}</span>
                <span>{'Verified Safe' if security_info['is_safe'] else 'Caution Advised'}</span>
            </div>
            {f"<div class="badge success">
                <span>✓</span>
                <span>Domain Age: {security_info['domain_age']} days</span>
            </div>" if security_info.get('domain_age') else ''}
        </div>

        <div>
            <button class="continue-btn" onclick="continueToSite()">
                Continue to Site
            </button>
            <button class="back-btn" onclick="goBack()">
                Go Back
            </button>
        </div>

        <div class="countdown" id="countdown">
            Auto-redirecting in <span id="seconds">5</span> seconds...
        </div>

        <a href="/report/{short_code}" class="report-link">
            🚩 Report suspicious link
        </a>
    </div>

    <script>
        let seconds = 5;
        const countdownElement = document.getElementById('seconds');

        const countdown = setInterval(() => {{
            seconds--;
            countdownElement.textContent = seconds;

            if (seconds <= 0) {{
                clearInterval(countdown);
                continueToSite();
            }}
        }}, 1000);

        function continueToSite() {{
```

```
            window.location.href = "{url_data.original_url}";
        }}

        function goBack() {{
            window.history.back();
        }}
    </script>
</body>
</html>
"""

return HTMLResponse(content=html_content)
```

Backend: routes/qr.py (QR Code Creation)

python

```python
from fastapi import APIRouter, HTTPException, Depends
from pydantic import BaseModel, HttpUrl
from services.qr_generator import generate_qr_code
from services.security import validate_url
from database import get_db
from models import URLs
import random
import string
from datetime import datetime

router = APIRouter()

class QRCreateRequest(BaseModel):
    url: HttpUrl
    show_preview: bool = True
    analytics_enabled: bool = True
    custom_title: str = None
    expiration_date: datetime = None

@router.post("/create")
async def create_qr(request: QRCreateRequest, user_id: str = Depends(get_current_user)):
    """
    Create new QR code
    """
    db = get_db()

    # 1. Validate URL security
    security_check = validate_url(str(request.url))

    if not security_check['is_safe']:
        raise HTTPException(
            status_code=400,
            detail=f"URL failed security check: {security_check['reason']}"
        )

    # 2. Generate unique short code
    short_code = generate_short_code(db)

    # 3. Save to database
    new_url = URLs(
        user_id=user_id,
        short_code=short_code,
        original_url=str(request.url),
```

```
        show_preview=request.show_preview,
        analytics_enabled=request.analytics_enabled,
        custom_title=request.custom_title,
        expiration_date=request.expiration_date,
        is_active=True,
        created_at=datetime.now()
    )

    db.add(new_url)
    db.commit()

    # 4. Generate QR code image
    app_url = os.getenv("APP_URL", "http://localhost:8000")
    qr_url = f"{app_url}/{short_code}"
    qr_image_path = generate_qr_code(qr_url, short_code)

    # 5. Return response
    return {
        "success": True,
        "short_code": short_code,
        "short_url": qr_url,
        "destination": str(request.url),
        "qr_image": f"/qr_codes/{short_code}.png",
        "created_at": new_url.created_at
    }


def generate_short_code(db, length=6):
    """
    Generate unique random short code
    """
    characters = string.ascii_letters + string.digits

    while True:
        short_code = ''.join(random.choices(characters, k=length))

        # Check if already exists
        existing = db.query(URLs).filter(URLs.short_code == short_code).first()

        if not existing:
            return short_code


@router.get("/list")
```

```python
async def list_qr_codes(user_id: str = Depends(get_current_user)):
    """
    Get all QR codes for current user
    """
    db = get_db()

    urls = db.query(URLs).filter(URLs.user_id == user_id).order_by(URLs.created_at.desc()).all()

    return {
        "success": True,
        "count": len(urls),
        "qr_codes": [
            {
                "short_code": url.short_code,
                "short_url": f"{os.getenv('APP_URL')}/{url.short_code}",
                "destination": url.original_url,
                "total_scans": url.total_scans,
                "is_active": url.is_active,
                "created_at": url.created_at,
                "qr_image": f"/qr_codes/{url.short_code}.png"
            }
            for url in urls
        ]
    }


@router.put("/{short_code}")
async def update_qr_destination(
    short_code: str,
    new_url: HttpUrl,
    user_id: str = Depends(get_current_user)
):
    """
    Update destination URL (dynamic QR feature)
    """
    db = get_db()

    # 1. Find QR code
    url_data = db.query(URLs).filter(
        URLs.short_code == short_code,
        URLs.user_id == user_id
    ).first()

    if not url_data:
```

```
        raise HTTPException(status_code=404, detail="QR code not found")

    # 2. Validate new URL
    security_check = validate_url(str(new_url))

    if not security_check['is_safe']:
        raise HTTPException(
            status_code=400,
            detail=f"URL failed security check: {security_check['reason']}"
        )

    # 3. Save old URL to history
    from models import URLHistory
    history = URLHistory(
        url_id=url_data.url_id,
        old_url=url_data.original_url,
        new_url=str(new_url),
        changed_at=datetime.now()
    )
    db.add(history)

    # 4. Update URL
    url_data.original_url = str(new_url)
    url_data.updated_at = datetime.now()

    db.commit()

    return {
        "success": True,
        "message": "Destination URL updated successfully",
        "short_code": short_code,
        "new_destination": str(new_url)
    }


@router.delete("/{short_code}")
async def delete_qr(short_code: str, user_id: str = Depends(get_current_user)):
    """
    Delete QR code (or deactivate)
    """
    db = get_db()

    url_data = db.query(URLs).filter(
        URLs.short_code == short_code,
```

```
    URLs.user_id == user_id
).first()

if not url_data:
    raise HTTPException(status_code=404, detail="QR code not found")

# Option 1: Soft delete (deactivate)
url_data.is_active = False
db.commit()

# Option 2: Hard delete (uncomment if preferred)
# db.delete(url_data)
# db.commit()

return {
    "success": True,
    "message": "QR code deactivated successfully"
}
```

Backend: services/qr_generator.py

python

```python
import qrcode
from PIL import Image
import os

def generate_qr_code(url: str, short_code: str, logo_path: str = None):
    """
    Generate QR code image

    Args:
        url: The URL to encode in QR code
        short_code: Unique identifier for filename
        logo_path: Optional path to logo image to embed

    Returns:
        Path to saved QR code image
    """
    # Create QR code instance
    qr = qrcode.QRCode(
        version=1,  # Controls size (1-40)
        error_correction=qrcode.constants.ERROR_CORRECT_H,  # High error correction (30%)
        box_size=10,  # Size of each box in pixels
        border=4,  # Border size in boxes
    )

    # Add data
    qr.add_data(url)
    qr.make(fit=True)

    # Create image
    img = qr.make_image(fill_color="black", back_color="white")

    # Embed logo if provided
    if logo_path and os.path.exists(logo_path):
        img = img.convert('RGB')
        logo = Image.open(logo_path)

        # Calculate logo size (10% of QR code size)
        qr_width, qr_height = img.size
        logo_size = int(qr_width / 10)
        logo = logo.resize((logo_size, logo_size), Image.LANCZOS)

        # Calculate position (center)
        logo_pos = ((qr_width - logo_size) // 2, (qr_height - logo_size) // 2)
```

```
    # Paste logo
    img.paste(logo, logo_pos)

# Create directory if doesn't exist
os.makedirs("qr_codes", exist_ok=True)

# Save image
file_path = f"qr_codes/{short_code}.png"
img.save(file_path)

return file_path


def generate_custom_qr(url: str, short_code: str, options: dict):
    """
    Generate QR code with custom styling

    Options:
        - fill_color: QR code color (hex)
        - back_color: Background color (hex)
        - logo_path: Path to logo image
        - style: 'square', 'rounded', 'dots'
    """
    qr = qrcode.QRCode(
        version=1,
        error_correction=qrcode.constants.ERROR_CORRECT_H,
        box_size=10,
        border=4,
    )

    qr.add_data(url)
    qr.make(fit=True)

    # Apply custom colors
    fill_color = options.get('fill_color', 'black')
    back_color = options.get('back_color', 'white')

    img = qr.make_image(fill_color=fill_color, back_color=back_color)

    # Apply logo if provided
    if 'logo_path' in options:
        img = img.convert('RGB')
        logo = Image.open(options['logo_path'])
```

```
    qr_width, qr_height = img.size
    logo_size = int(qr_width / 10)
    logo = logo.resize((logo_size, logo_size), Image.LANCZOS)

    logo_pos = ((qr_width - logo_size) // 2, (qr_height - logo_size) // 2)
    img.paste(logo, logo_pos)

# Save
file_path = f"qr_codes/{short_code}.png"
img.save(file_path)

return file_path
```

Backend: services/security.py

python

```python
import requests
import ssl
import socket
from urllib.parse import urlparse
import os
import whois
from datetime import datetime

def validate_url(url: str) -> dict:
    """
    Comprehensive URL security validation

    Returns dict with:
        - is_safe (bool)
        - reason (str)
        - details (dict)
    """
    result = {
        "is_safe": True,
        "reason": "",
        "details": {}
    }

    # 1. Check URL format
    try:
        parsed = urlparse(url)
        if not parsed.scheme or not parsed.netloc:
            result["is_safe"] = False
            result["reason"] = "Invalid URL format"
            return result
    except:
        result["is_safe"] = False
        result["reason"] = "Malformed URL"
        return result

    # 2. Check if URL is reachable
    try:
        response = requests.head(url, timeout=5, allow_redirects=True)
        result["details"]["status_code"] = response.status_code

        if response.status_code >= 400:
            result["is_safe"] = False
            result["reason"] = f"URL not reachable (HTTP {response.status_code})"
```

```
            return result
    except requests.exceptions.RequestException as e:
        result["is_safe"] = False
        result["reason"] = f"Cannot connect to URL: {str(e)}"
        return result

    # 3. Check Google Safe Browsing
    safe_browsing_result = check_safe_browsing(url)
    result["details"]["safe_browsing"] = safe_browsing_result

    if not safe_browsing_result["is_safe"]:
        result["is_safe"] = False
        result["reason"] = "URL flagged by Google Safe Browsing"
        return result

    # 4. Check SSL certificate
    ssl_result = check_ssl_certificate(url)
    result["details"]["ssl"] = ssl_result

    # 5. Check domain age (warn if < 30 days)
    domain_age = check_domain_age(url)
    result["details"]["domain_age"] = domain_age

    if domain_age.get("age_days") and domain_age["age_days"] < 30:
        # Don't block, but flag as suspicious
        result["details"]["warning"] = "Domain is less than 30 days old"

    return result


def check_safe_browsing(url: str) -> dict:
    """
    Check URL against Google Safe Browsing API
    """
    api_key = os.getenv("GOOGLE_SAFE_BROWSING_KEY")

    if not api_key:
        return {"is_safe": True, "note": "API key not configured"}

    endpoint = "https://safebrowsing.googleapis.com/v4/threatMatches:find"

    payload = {
        "client": {
            "clientId": "qrsecure",
```

```
            "clientVersion": "1.0.0"
        },
        "threatInfo": {
            "threatTypes": ["MALWARE", "SOCIAL_ENGINEERING", "UNWANTED_SOFTWARE", "POTENTIALLY_HAR
            "platformTypes": ["ANY_PLATFORM"],
            "threatEntryTypes": ["URL"],
            "threatEntries": [{"url": url}]
        }
    }

    try:
        response = requests.post(
            f"{endpoint}?key={api_key}",
            json=payload,
            timeout=5
        )

        data = response.json()

        if "matches" in data and len(data["matches"]) > 0:
            return {
                "is_safe": False,
                "threats": [match["threatType"] for match in data["matches"]]
            }

        return {"is_safe": True}

    except Exception as e:
        # If API fails, don't block the URL
        return {"is_safe": True, "error": str(e)}


def check_ssl_certificate(url: str) -> dict:
    """
    Check if URL has valid SSL certificate
    """
    parsed = urlparse(url)
    domain = parsed.netloc

    # Only check HTTPS URLs
    if parsed.scheme != "https":
        return {"has_ssl": False, "reason": "Not HTTPS"}

    try:
```

```
        context = ssl.create_default_context()
        with socket.create_connection((domain, 443), timeout=5) as sock:
            with context.wrap_socket(sock, server_hostname=domain) as ssock:
                cert = ssock.getpeercert()

                # Extract certificate details
                not_after = cert.get('notAfter')
                issuer = dict(x[0] for x in cert.get('issuer', []))

                return {
                    "has_ssl": True,
                    "issuer": issuer.get('organizationName', 'Unknown'),
                    "valid_until": not_after
                }
    except Exception as e:
        return {
            "has_ssl": False,
            "error": str(e)
        }


def check_domain_age(url: str) -> dict:
    """
    Check domain registration age
    """
    parsed = urlparse(url)
    domain = parsed.netloc

    try:
        w = whois.whois(domain)
        creation_date = w.creation_date

        # Handle list of dates (some domains have multiple)
        if isinstance(creation_date, list):
            creation_date = creation_date[0]

        if creation_date:
            age_days = (datetime.now() - creation_date).days

            return {
                "age_days": age_days,
                "created": creation_date.strftime("%Y-%m-%d"),
                "is_new": age_days < 30
            }
```

```
    except Exception as e:
        return {
            "age_days": None,
            "error": str(e)
        }


def get_security_info(url: str) -> dict:
    """
    Get all security information for preview page
    """
    ssl_info = check_ssl_certificate(url)
    safe_browsing = check_safe_browsing(url)
    domain_age = check_domain_age(url)

    return {
        "has_ssl": ssl_info.get("has_ssl", False),
        "is_safe": safe_browsing.get("is_safe", True),
        "domain_age": domain_age.get("age_days"),
        "created_date": domain_age.get("created")
    }
```

Backend: services/analytics.py

python

```python
from database import get_db
from models import Scans
from services.geolocation import get_location_from_ip
from user_agents import parse
from datetime import datetime, timedelta

async def log_scan(url_id: str, ip_address: str, user_agent: str, db):
    """
    Log a QR code scan with analytics data
    """
    # Parse user agent
    ua = parse(user_agent)

    # Get location from IP
    location = get_location_from_ip(ip_address)

    # Create scan record
    scan = Scans(
        url_id=url_id,
        scanned_at=datetime.now(),
        ip_address=ip_address,
        country=location.get("country", "Unknown"),
        city=location.get("city", "Unknown"),
        device_type=ua.device.family,
        os=f"{ua.os.family} {ua.os.version_string}",
        browser=f"{ua.browser.family} {ua.browser.version_string}",
        user_agent=user_agent
    )

    db.add(scan)

    # Update total scans counter
    from models import URLs
    url_data = db.query(URLs).filter(URLs.url_id == url_id).first()
    if url_data:
        url_data.total_scans += 1

    db.commit()


def get_analytics_summary(url_id: str, days: int = 30):
    """
    Get analytics summary for a QR code
```

```
    """
    db = get_db()

    # Get scans from last N days
    cutoff_date = datetime.now() - timedelta(days=days)
    scans = db.query(Scans).filter(
        Scans.url_id == url_id,
        Scans.scanned_at >= cutoff_date
    ).all()

    # Calculate metrics
    total_scans = len(scans)
    scans_today = len([s for s in scans if s.scanned_at.date() == datetime.now().date()])

    # Group by location
    locations = {}
    for scan in scans:
        key = f"{scan.city}, {scan.country}"
        locations[key] = locations.get(key, 0) + 1

    top_locations = sorted(locations.items(), key=lambda x: x[1], reverse=True)[:10]

    # Group by device
    devices = {}
    for scan in scans:
        devices[scan.device_type] = devices.get(scan.device_type, 0) + 1

    # Group by date (timeline)
    timeline = {}
    for scan in scans:
        date_key = scan.scanned_at.strftime("%Y-%m-%d")
        timeline[date_key] = timeline.get(date_key, 0) + 1

    timeline_sorted = sorted(timeline.items())

    return {
        "total_scans": total_scans,
        "scans_today": scans_today,
        "top_locations": top_locations,
        "devices": devices,
        "timeline": timeline_sorted
    }
```

Backend: services/geolocation.py

python

```
import geoip2.database
import os

# Load GeoLite2 database
reader = None

def initialize_geoip():
    global reader
    db_path = "GeoLite2-City.mmdb"

    if os.path.exists
```