```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>BookTracker - Scan & Track Your Reading</title>
    <meta name="theme-color" content="#cc5500">
    <meta name="description" content="Scan, track, and review your reading journey">

    <!-- PWA Manifest -->
    <link rel="manifest" href="manifest.json">

    <!-- iOS Support -->
    <meta name="apple-mobile-web-app-capable" content="yes">
    <meta name="apple-mobile-web-app-status-bar-style" content="black-translucent">
    <meta name="apple-mobile-web-app-title" content="BookTracker">
    <link rel="apple-touch-icon" href="icon-192.png">

    <!-- Favicon -->
    <link rel="icon" type="image/png" sizes="192x192" href="icon-192.png">
    <link rel="icon" type="image/png" sizes="512x512" href="icon-512.png">

    <style>
        * { margin: 0; padding: 0; box-sizing: border-box; }
        body {
            font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif;
            background: linear-gradient(135deg, #cc5500 0%, #1a1a1a 100%);
            min-height: 100vh; color: #fff; font-weight: bold;
        }
        .container {
            max-width: 500px; margin: 0 auto; background: #2a2a2a;
            min-height: 100vh; box-shadow: 0 0 20px rgba(0,0,0,0.5);
        }
        .header {
            background: linear-gradient(135deg, #cc5500 0%, #1a1a1a 100%);
            color: white; padding: 20px; text-align: center;
        }
        .nav-tabs { display: flex; background: #1a1a1a; }
        .nav-tab {
            flex: 1; padding: 15px; text-align: center; background: none;
            border: none; cursor: pointer; font-size: 14px; font-weight: bold;
            color: #fff;
        }
        .nav-tab.active { background: #cc5500; color: white; border-bottom: 2px solid #fff; }
```

```
.tab-content { display: none; padding: 20px; min-height: calc(100vh - 160px); }
.tab-content.active { display: block; }
.scan-btn {
    background: linear-gradient(135deg, #cc5500 0%, #1a1a1a 100%);
    color: white; border: none; padding: 20px 40px; border-radius: 50px;
    font-size: 18px; font-weight: bold; cursor: pointer; margin: 20px 0;
    box-shadow: 0 4px 15px rgba(204, 85, 0, 0.4); width: 100%; max-width: 300px;
}
.camera-container {
    position: relative; margin: 20px auto; max-width: 300px;
    border-radius: 10px; overflow: hidden; box-shadow: 0 4px 15px rgba(0,0,0,0.2);
}
#video { width: 100%; height: auto; display: block; }

/* Enhanced scan overlay with animations */
.scan-overlay {
    position: absolute; top: 50%; left: 50%; transform: translate(-50%, -50%);
    width: 200px; height: 100px; border: 2px solid #cc5500; border-radius: 8px;
    box-shadow: 0 0 0 9999px rgba(0,0,0,0.7);
    transition: all 0.3s ease;
}
.scan-overlay::before {
    content: '';
    position: absolute;
    top: -10px; left: -10px; right: -10px; bottom: -10px;
    border: 2px solid transparent;
    border-radius: 12px;
    animation: scanPulse 2s infinite;
}
@keyframes scanPulse {
    0% { border-color: transparent; }
    50% { border-color: rgba(204, 85, 0, 0.5); }
    100% { border-color: transparent; }
}

/* Scanning status indicator */
.scan-status {
    position: absolute;
    bottom: 20px;
    left: 50%;
    transform: translateX(-50%);
    background: rgba(0, 0, 0, 0.8);
    color: white;
    padding: 8px 16px;
```

```
        border-radius: 20px;
        font-size: 14px;
        font-weight: bold;
    }

    .book-card {
        background: #333333; border-radius: 15px; padding: 20px; margin-bottom: 20px;
        box-shadow: 0 4px 15px rgba(0,0,0,0.3); color: white;
    }
    .book-info { display: flex; gap: 15px; margin-bottom: 15px; }
    .book-cover {
        width: 80px; height: 120px; object-fit: cover;
        border-radius: 8px; background: #1a1a1a;
    }
    .book-details h3 { font-size: 16px; margin-bottom: 5px; color: white; font-weight: bold; }
    .book-details p { font-size: 14px; color: #ccc; margin-bottom: 3px; font-weight: bold; }
    .btn {
        padding: 8px 16px; border: none; border-radius: 20px;
        font-size: 12px; font-weight: bold; cursor: pointer; margin: 2px;
    }
    .btn-primary { background: #cc5500; color: white; }
    .btn-success { background: #cc5500; color: white; }
    .btn-warning { background: #666; color: white; }
    .btn-danger { background: #1a1a1a; color: white; }
    .progress-bar {
        width: 100%; height: 8px; background: #1a1a1a;
        border-radius: 4px; margin: 10px 0;
    }
    .progress-fill {
        height: 100%; background: linear-gradient(90deg, #cc5500, #ff6600);
        border-radius: 4px; transition: width 0.3s ease;
    }
    .notes-area {
        width: 100%; min-height: 80px; padding: 10px;
        border: 1px solid #555; border-radius: 8px; margin: 10px 0;
        background: #1a1a1a; color: white; font-weight: bold;
    }
    .page-input { display: flex; align-items: center; gap: 10px; margin: 10px 0; color: white;
font-weight: bold; }
    .page-input input {
        width: 80px; padding: 5px 10px; border: 1px solid #555;
        border-radius: 4px; text-align: center; background: #1a1a1a; color: white; font-weight:
bold;
    }
```

```
.status-badges { display: flex; gap: 5px; margin: 10px 0; }
.badge {
    padding: 4px 8px; border-radius: 12px; font-size: 11px; font-weight: bold;
}
.badge-reading { background: #cc5500; color: white; }
.badge-finished { background: #cc5500; color: white; }
.badge-dnf { background: #666; color: white; }
.badge-recommend { background: #cc5500; color: white; }
.badge-not-recommend { background: #1a1a1a; color: white; }
.empty-state { text-align: center; padding: 60px 20px; color: white; font-weight: bold; }
.loading { text-align: center; padding: 40px; color: white; font-weight: bold; }
.spinner {
    width: 40px; height: 40px; border: 4px solid #333;
    border-top: 4px solid #cc5500; border-radius: 50%;
    animation: spin 1s linear infinite; margin: 0 auto 20px;
}
@keyframes spin { 0% { transform: rotate(0deg); } 100% { transform: rotate(360deg); } }
.error {
    background: #1a1a1a; color: #cc5500; padding: 15px;
    border-radius: 8px; margin: 20px 0; text-align: center; font-weight: bold;
    border: 1px solid #cc5500;
}
.book-actions { display: flex; gap: 5px; flex-wrap: wrap; }

/* PWA Install Banner */
.install-banner {
    position: fixed; bottom: 20px; left: 20px; right: 20px;
    background: #cc5500; color: white; padding: 15px; border-radius: 10px;
    display: none; align-items: center; justify-content: space-between;
    box-shadow: 0 4px 15px rgba(0,0,0,0.3); z-index: 1000;
}
.install-banner button {
    background: white; color: #cc5500; border: none;
    padding: 8px 16px; border-radius: 5px; font-weight: bold; cursor: pointer;
}

/* Camera permission info */
.camera-info {
    background: #333; padding: 15px; border-radius: 8px; margin: 20px 0;
    color: #ccc; font-size: 14px; text-align: center;
}

/* Manual ISBN input styles */
.manual-input-section {
```

```
            background: #333; padding: 20px; border-radius: 8px; margin: 20px 0;
            text-align: center;
        }
        .manual-input-section h3 {
            color: #cc5500; margin-bottom: 15px;
        }
        .isbn-input {
            width: 100%; max-width: 200px; padding: 10px; margin: 10px 0;
            border: 1px solid #555; border-radius: 4px; text-align: center;
            background: #1a1a1a; color: white; font-weight: bold; font-size: 16px;
        }
        .scanned-info {
            background: #1a1a1a; padding: 10px; border-radius: 8px; margin: 10px 0;
            font-size: 12px; color: #ccc; text-align: center;
        }

        @media (max-width: 480px) {
            .book-info { flex-direction: column; align-items: center; text-align: center; }
            .book-actions { justify-content: center; }
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h1>📚 BookTracker</h1>
            <p>Scan, track, and review your reading journey</p>
        </div>

        <div class="nav-tabs">
            <button class="nav-tab active" data-tab="scan">📱 Scan</button>
            <button class="nav-tab" data-tab="library">📖 Library</button>
        </div>

        <div id="scan-tab" class="tab-content active">
            <div style="text-align: center; padding: 40px 20px;">
                <button class="scan-btn" id="scan-button">📷 Scan Book Barcode</button>
                <p style="color: #666; font-size: 14px;">Point your camera at a book's barcode</p>

                <div class="camera-info" id="camera-info" style="display: none;">
                    <p>📱 Please allow camera access when prompted</p>
                    <p>This lets the app scan book barcodes</p>
                </div>
```

```
            <div id="camera-container" class="camera-container" style="display: none;">
                <video id="video" autoplay muted playsinline></video>
                <div class="scan-overlay"></div>
                <div class="scan-status">🔍 Point camera at barcode</div>
            </div>

            <button id="stop-btn" class="btn btn-danger" style="display: none; margin-top:
10px;">Stop Scanning</button>
            <button id="focus-btn" class="btn btn-warning" style="display: none; margin-top:
10px;">🎯 Focus</button>
            <button id="refresh-btn" class="btn btn-warning" style="display: none; margin-top:
10px;">🔄 Refresh Camera</button>

            <!-- Manual ISBN input section -->
            <div class="manual-input-section">
                <h3>📝 Manual ISBN Entry</h3>
                <p style="color: #ccc; font-size: 12px; margin-bottom: 10px;">Enter ISBN if
scanning doesn't work</p>
                <input type="text" id="manual-isbn" class="isbn-input" placeholder="Enter ISBN..."
maxlength="13">
                <br>
                <button class="btn btn-primary" id="manual-lookup-btn">🔍 Look Up
Book</button>
            </div>

            <div id="scan-result" style="display: none;">
                <div class="loading">
                    <div class="spinner"></div>
                    <p>Looking up book information...</p>
                </div>
            </div>
        </div>
    </div>

    <div id="library-tab" class="tab-content">
        <div id="library-content">
            <div class="empty-state">
                <h3>Your library is empty</h3>
                <p>Start by scanning your first book!</p>
            </div>
        </div>
    </div>
  </div>
```

```
    <!-- PWA Install Banner -->
    <div id="install-banner" class="install-banner">
        <span>📱 Install BookTracker for easier access!</span>
        <button id="install-btn">Install</button>
        <button id="dismiss-install" style="background: transparent; color: white;">✕</button>
    </div>

    <!-- Single jsQR loader with fallbacks -->
    <script>
        // Unified jsQR loader with multiple CDN fallbacks
        (function loadJsQR() {
            const cdnList = [
                'https://cdnjs.cloudflare.com/ajax/libs/jsqr/1.4.0/jsQR.min.js',
                'https://unpkg.com/jsqr@1.4.0/dist/jsQR.js',
                'https://cdn.jsdelivr.net/npm/jsqr@1.4.0/dist/jsQR.js'
            ];

            let currentCdnIndex = 0;
            let loadAttempts = 0;
            const maxAttempts = 3;

            function tryLoadCdn() {
                if (typeof jsQR !== 'undefined') {
                    console.log('✅ jsQR already loaded');
                    return;
                }

                if (loadAttempts >= maxAttempts) {
                    console.error('❌ All jsQR CDNs failed to load');
                    showError('Barcode scanner library failed to load. Please refresh and try again.');
                    return;
                }

                const script = document.createElement('script');
                script.src = cdnList[currentCdnIndex];

                script.onload = function() {
                    console.log(`✅ jsQR loaded from CDN ${currentCdnIndex + 1}:
${cdnList[currentCdnIndex]}`);
                };

                script.onerror = function() {
                    console.log(`❌ CDN ${currentCdnIndex + 1} failed: ${cdnList[currentCdnIndex]}`);
                    currentCdnIndex = (currentCdnIndex + 1) % cdnList.length;
```

```
            loadAttempts++;
            setTimeout(tryLoadCdn, 1000);
        };

        document.head.appendChild(script);
    }

    tryLoadCdn();
  })();
</script>

<script>
  let currentStream = null;
  let scanningActive = false;
  let books = [];
  let deferredPrompt;
  let lastScannedBarcode = null;
  let lastScanTime = 0;
  let scanCanvas = null; // Reuse canvas to prevent memory leaks

  // Debug logging function
  function debugLog(message) {
    console.log('BookTracker:', message);
  }

  // Reset scanning state function
  function resetScanningState() {
    lastScannedBarcode = null;
    lastScanTime = 0;
    console.log('🧹 Scanning state reset');
  }

  // Camera refocus function
  function requestCameraFocus() {
    if (currentStream) {
      const videoTrack = currentStream.getVideoTracks()[0];
      if (videoTrack && videoTrack.getCapabilities) {
        try {
          const capabilities = videoTrack.getCapabilities();
          if (capabilities.focusMode && capabilities.focusMode.includes('continuous')) {
            videoTrack.applyConstraints({
              advanced: [{ focusMode: 'continuous' }]
            }).catch(error => {
              debugLog('Focus constraint failed: ' + error.message);
```

```
                });
            }
        } catch (error) {
            debugLog('Focus capability check failed: ' + error.message);
        }
    }
}
}

// Refresh camera stream function
function refreshCameraStream() {
    if (currentStream) {
        debugLog('Refreshing camera stream...');
        stopScanning();
        setTimeout(() => {
            startScanning();
        }, 500);
    }
}

// Show scan success feedback
function showScanSuccess() {
    const overlay = document.querySelector('.scan-overlay');
    if (overlay) {
        overlay.style.borderColor = '#00ff00';
        overlay.style.boxShadow = '0 0 20px #00ff00, 0 0 0 9999px rgba(0,0,0,0.7)';
        setTimeout(() => {
            overlay.style.borderColor = '#cc5500';
            overlay.style.boxShadow = '0 0 0 9999px rgba(0,0,0,0.7)';
        }, 1000);
    }

    // Audio feedback
    try {
        const audioContext = new (window.AudioContext || window.webkitAudioContext)();
        const oscillator = audioContext.createOscillator();
        const gainNode = audioContext.createGain();

        oscillator.connect(gainNode);
        gainNode.connect(audioContext.destination);

        oscillator.frequency.setValueAtTime(800, audioContext.currentTime);
        oscillator.frequency.setValueAtTime(600, audioContext.currentTime + 0.1);
        gainNode.gain.setValueAtTime(0.3, audioContext.currentTime);
```

```
            gainNode.gain.exponentialRampToValueAtTime(0.01, audioContext.currentTime +
0.2);

            oscillator.start(audioContext.currentTime);
            oscillator.stop(audioContext.currentTime + 0.2);
        } catch (error) {
            debugLog('Audio feedback failed: ' + error.message);
        }
    }

    // Check camera support on load
    function checkCameraSupport() {
        if (!navigator.mediaDevices || !navigator.mediaDevices.getUserMedia) {
            showError('Camera not supported in this browser. Please use Chrome, Firefox, or
Safari.');
            return false;
        }
        if (location.protocol !== 'https:' && location.hostname !== 'localhost') {
            showError('Camera requires HTTPS. Please access this app over HTTPS.');
            return false;
        }
        return true;
    }

    // PWA Install Logic
    window.addEventListener('beforeinstallprompt', (e) => {
        e.preventDefault();
        deferredPrompt = e;
        document.getElementById('install-banner').style.display = 'flex';
    });

    document.getElementById('install-btn').addEventListener('click', async () => {
        if (deferredPrompt) {
            deferredPrompt.prompt();
            const { outcome } = await deferredPrompt.userChoice;
            debugLog('Install prompt result: ' + outcome);
            deferredPrompt = null;
            document.getElementById('install-banner').style.display = 'none';
        }
    });

    document.getElementById('dismiss-install').addEventListener('click', () => {
        document.getElementById('install-banner').style.display = 'none';
    });
```

```
// Register Service Worker
if ('serviceWorker' in navigator) {
    window.addEventListener('load', () => {
        navigator.serviceWorker.register('./sw.js')
            .then(registration => {
                debugLog('SW registered');
            })
            .catch(registrationError => {
                debugLog('SW registration failed: ' + registrationError);
            });
    });
}

// Safe storage functions
function getStoredBooks() {
    try {
        return JSON.parse(localStorage.getItem('books') || '[]');
    } catch (e) {
        debugLog('localStorage not available');
        return [];
    }
}

function saveBooks() {
    try {
        localStorage.setItem('books', JSON.stringify(books));
    } catch (e) {
        debugLog('localStorage save failed');
    }
}

// Initialize app
document.addEventListener('DOMContentLoaded', function() {
    debugLog('App initializing...');

    if (!checkCameraSupport()) {
        return;
    }

    books = getStoredBooks();
    renderLibrary();

    // Navigation
```

```
    document.addEventListener('click', function(e) {
        if (e.target.classList.contains('nav-tab')) {
            switchTab(e.target.getAttribute('data-tab'));
        }
    });

    // Button event listeners
    const scanButton = document.getElementById('scan-button');
    const stopButton = document.getElementById('stop-btn');
    const focusButton = document.getElementById('focus-btn');
    const refreshButton = document.getElementById('refresh-btn');
    const manualLookupBtn = document.getElementById('manual-lookup-btn');
    const manualIsbnInput = document.getElementById('manual-isbn');

    if (scanButton) {
        scanButton.addEventListener('click', handleScanButtonClick);
    }
    if (stopButton) {
        stopButton.addEventListener('click', handleStopButtonClick);
    }
    if (focusButton) {
        focusButton.addEventListener('click', requestCameraFocus);
    }
    if (refreshButton) {
        refreshButton.addEventListener('click', refreshCameraStream);
    }
    if (manualLookupBtn) {
        manualLookupBtn.addEventListener('click', handleManualLookup);
    }
    if (manualIsbnInput) {
        manualIsbnInput.addEventListener('keypress', function(e) {
            if (e.key === 'Enter') {
                handleManualLookup();
            }
        });
        manualIsbnInput.addEventListener('input', function(e) {
            let value = e.target.value.replace(/[^\d\-X]/gi, '');
            e.target.value = value;
        });
    }

    debugLog('App initialized successfully');
});
```

```
// Handle manual ISBN lookup
async function handleManualLookup() {
    const isbnInput = document.getElementById('manual-isbn');
    const rawIsbn = isbnInput.value.trim();

    if (!rawIsbn) {
        showError('Please enter an ISBN number');
        return;
    }

    const isbn = rawIsbn.replace(/\D/g, '');
    if (isbn.length !== 10 && isbn.length !== 13) {
        showError('Please enter a valid 10 or 13 digit ISBN');
        return;
    }

    debugLog('Manual ISBN lookup: ' + isbn);

    document.getElementById('scan-result').style.display = 'block';
    document.getElementById('scan-result').innerHTML = `
        <div class="loading">
            <div class="spinner"></div>
            <p>Looking up book information...</p>
            <div class="scanned-info">Manual lookup: ${isbn}</div>
        </div>
    `;

    try {
        await lookupBookByISBN(isbn, 'manual');
    } catch (error) {
        debugLog('Manual lookup error: ' + error.message);
        showError('Error looking up book. Check internet connection.');
        document.getElementById('scan-result').style.display = 'none';
    }
}

// Button handlers
async function handleScanButtonClick(e) {
    e.preventDefault();
    debugLog('Scan button clicked - starting camera...');

    if (typeof jsQR === 'undefined') {
        showError('Barcode scanner is still loading. Please wait a moment and try again.');
        return;
```

```
        }

        document.getElementById('camera-info').style.display = 'block';
        setTimeout(() => {
            startScanning();
        }, 500);
    }

    function handleStopButtonClick(e) {
        e.preventDefault();
        debugLog('Stop button clicked');
        stopScanning();
    }

    function switchTab(tabName) {
        document.querySelectorAll('.nav-tab').forEach(tab => tab.classList.remove('active'));
        document.querySelector(`[data-tab="${tabName}"]`).classList.add('active');

        document.querySelectorAll('.tab-content').forEach(content =>
content.classList.remove('active'));
        document.getElementById(tabName + '-tab').classList.add('active');

        if (tabName === 'library') renderLibrary();
    }

    async function startScanning() {
        resetScanningState();
        debugLog('Starting camera access...');

        try {
            document.getElementById('camera-info').innerHTML = '<p>🔄 Requesting camera
access...</p>';

            let stream = null;
            const constraints = [
                {
                    video: {
                        facingMode: { ideal: 'environment' },
                        width: { ideal: 1280, max: 1920 },
                        height: { ideal: 720, max: 1080 }
                    }
                },
                {
                    video: {
```

```
            facingMode: { ideal: 'environment' }
        }
    },
    {
        video: true
    }
];

for (let i = 0; i < constraints.length; i++) {
    try {
        debugLog(`Trying camera constraint ${i + 1}...`);
        stream = await navigator.mediaDevices.getUserMedia(constraints[i]);
        debugLog(`Camera constraint ${i + 1} successful`);
        break;
    } catch (err) {
        debugLog(`Camera constraint ${i + 1} failed: ${err.name}`);
        if (i === constraints.length - 1) {
            throw err;
        }
    }
}

if (!stream) {
    throw new Error('No camera stream obtained');
}

currentStream = stream;
debugLog('Camera access granted successfully');

const video = document.getElementById('video');
if (!video) {
    throw new Error('Video element not found');
}

video.srcObject = currentStream;
requestCameraFocus();

document.getElementById('camera-info').style.display = 'none';
document.getElementById('camera-container').style.display = 'block';
document.getElementById('stop-btn').style.display = 'block';
document.getElementById('focus-btn').style.display = 'block';
document.getElementById('refresh-btn').style.display = 'block';
document.getElementById('scan-button').style.display = 'none';
```

```
            video.addEventListener('loadedmetadata', function() {
                debugLog('Video metadata loaded, starting barcode scanning...');
                scanningActive = true;
                initializeScanCanvas();
                scanForBarcode();
            }, { once: true });

            video.addEventListener('canplay', function() {
                if (!scanningActive) {
                    debugLog('Video can play, starting barcode scanning...');
                    scanningActive = true;
                    initializeScanCanvas();
                    scanForBarcode();
                }
            }, { once: true });

        } catch (error) {
            debugLog('Camera error occurred: ' + error.name + ' - ' + error.message);
            document.getElementById('camera-info').style.display = 'none';

            if (error.name === 'NotAllowedError') {
                showError('❌ Camera permission denied. Please click "Allow" when prompted and
try again.');
            } else if (error.name === 'NotFoundError') {
                showError('❌ No camera found on this device.');
            } else if (error.name === 'NotSupportedError') {
                showError('❌ Camera not supported on this device.');
            } else if (error.name === 'NotReadableError') {
                showError('❌ Camera is being used by another application.');
            } else if (error.name === 'OverconstrainedError') {
                showError('❌ Camera constraints could not be satisfied.');
            } else if (error.name === 'SecurityError') {
                showError('❌ Camera access blocked by security policy. Please use HTTPS.');
            } else {
                showError('❌ Camera error: ' + error.message);
            }
        }
    }

    function stopScanning() {
        debugLog('Stopping camera...');
        scanningActive = false;

        if (currentStream) {
```

```
        currentStream.getTracks().forEach(track => {
            track.stop();
            debugLog('Camera track stopped');
        });
        currentStream = null;
    }

    // Clean up canvas
    if (scanCanvas) {
        scanCanvas = null;
    }

    document.getElementById('camera-info').style.display = 'none';
    document.getElementById('camera-container').style.display = 'none';
    document.getElementById('stop-btn').style.display = 'none';
    document.getElementById('focus-btn').style.display = 'none';
    document.getElementById('refresh-btn').style.display = 'none';
    document.getElementById('scan-button').style.display = 'block';
    document.getElementById('scan-result').style.display = 'none';
}

// Initialize reusable canvas for scanning
function initializeScanCanvas() {
    if (!scanCanvas) {
        scanCanvas = document.createElement('canvas');
    }
}

function scanForBarcode() {
    if (!scanningActive) return;

    const video = document.getElementById('video');
    if (!video || video.readyState !== video.HAVE_ENOUGH_DATA) {
        requestAnimationFrame(scanForBarcode);
        return;
    }

    try {
        const ctx = scanCanvas.getContext('2d', { willReadFrequently: true });
        const width = video.videoWidth;
        const height = video.videoHeight;

        if (width === 0 || height === 0) {
            requestAnimationFrame(scanForBarcode);
```

```
        return;
    }

    scanCanvas.width = width;
    scanCanvas.height = height;

    ctx.clearRect(0, 0, width, height);
    ctx.drawImage(video, 0, 0, width, height);

    const imageData = ctx.getImageData(0, 0, width, height);

    if (!imageData || !imageData.data || imageData.data.length === 0) {
        requestAnimationFrame(scanForBarcode);
        return;
    }

    if (typeof jsQR !== 'undefined') {
        const qrResult = jsQR(imageData.data, width, height, {
            inversionAttempts: "both"
        });

        if (qrResult && qrResult.data && qrResult.data.trim() !== '') {
            const rawBarcode = qrResult.data.trim();
            console.log('🔍 RAW BARCODE DETECTED:', rawBarcode);

            const numericPart = rawBarcode.replace(/\D/g, '');
            if (numericPart.length >= 10 && numericPart.length <= 13) {
                console.log('✅ VALID BARCODE DETECTED:', rawBarcode);
                scanningActive = false;
                showScanSuccess();
                handleBarcodeDetected(rawBarcode);
                return;
            } else {
                console.log('⚠️ Invalid barcode length:', numericPart.length, 'digits');
            }
        }
    }
} catch (error) {
    console.error('Scan error:', error);
}

if (scanningActive) {
    requestAnimationFrame(scanForBarcode);
}
```

```
}

async function handleBarcodeDetected(barcode) {
    const cleanedISBN = barcode.replace(/\D/g, '');

    // Check for duplicate scans with shorter timeout
    if (lastScannedBarcode === barcode && Date.now() - lastScanTime < 2000) {
        debugLog('Duplicate barcode detected within 2 seconds, ignoring');
        setTimeout(() => {
            scanningActive = true;
            scanForBarcode();
        }, 500);
        return;
    }

    lastScannedBarcode = barcode;
    lastScanTime = Date.now();

    debugLog('Processing barcode: ' + barcode);
    console.log('✅ PROCESSING ISBN:', cleanedISBN);

    document.getElementById('scan-result').style.display = 'block';
    document.getElementById('scan-result').innerHTML = `
        <div class="loading">
            <div class="spinner"></div>
            <p>Looking up book information...</p>
            <div class="scanned-info">
                <strong>Scanned:</strong> ${barcode}<br>
                <strong>ISBN:</strong> ${cleanedISBN}<br>
                <strong>Length:</strong> ${cleanedISBN.length} digits
            </div>
        </div>
    `;

    try {
        await lookupBookByISBN(cleanedISBN, 'scanned', barcode);
    } catch (error) {
        debugLog('Book lookup error: ' + error.message);
        showError('Error looking up book. Check internet connection.');
        setTimeout(() => {
            document.getElementById('scan-result').style.display = 'none';
            scanningActive = true;
            scanForBarcode();
        }, 3000);
```

```
        }
    }

    async function lookupBookByISBN(isbn, source, originalBarcode = null) {
        debugLog(`Looking up ISBN: ${isbn}`);
        console.log('🔍 API LOOKUP - ISBN:', isbn, 'Source:', source, 'Original:',
originalBarcode);

        let bookData = null;
        const sources = [
            { name: 'Google Books', func: fetchBookFromGoogle },
            { name: 'OpenLibrary', func: fetchBookFromOpenLibrary }
        ];

        for (const sourceInfo of sources) {
            if (!bookData) {
                debugLog(`Trying ${sourceInfo.name}...`);
                try {
                    bookData = await sourceInfo.func(isbn);
                    if (bookData) {
                        debugLog(`✅ Found book in ${sourceInfo.name}`);
                        bookData.source = sourceInfo.name;
                        break;
                    }
                } catch (error) {
                    debugLog(`❌ ${sourceInfo.name} failed: ${error.message}`);
                }
            }
        }

        if (bookData) {
            bookData.scanInfo = {
                source: source,
                originalBarcode: originalBarcode,
                cleanedISBN: isbn,
                timestamp: new Date().toISOString(),
                foundIn: bookData.source
            };
            displayBookPreview(bookData);
        } else {
            const errorMsg = source === 'manual'
                ? `No book found for ISBN: ${isbn}. This might be a region-specific edition or the
ISBN might not be in our databases.`
                : 'Book not found in any database. Try manual entry or a different barcode.';
```

```
        showError(errorMsg);

        if (source === 'scanned') {
            setTimeout(() => {
                document.getElementById('scan-result').style.display = 'none';
                scanningActive = true;
                scanForBarcode();
            }, 4000);
        } else {
            document.getElementById('scan-result').style.display = 'none';
        }
    }
}

async function fetchBookFromGoogle(isbn) {
    try {
        const url = `https://www.googleapis.com/books/v1/volumes?q=isbn:${isbn}`;
        console.log('📡 Google Books API call:', url);

        const response = await fetch(url);
        const data = await response.json();
        console.log('📡 Google Books Response:', data);

        if (data.items && data.items.length > 0) {
            const book = data.items[0].volumeInfo;
            const result = {
                isbn: isbn,
                title: book.title || 'Unknown Title',
                authors: book.authors || ['Unknown Author'],
                pageCount: book.pageCount || 0,
                coverUrl: book.imageLinks?.thumbnail?.replace('http:', 'https:') || '',
                publishedDate: book.publishedDate || 'Unknown'
            };

            console.log('📚 Google Books Result:', result);
            return result;
        }
    } catch (error) {
        debugLog('Google Books API error: ' + error.message);
    }
    return null;
}
```

```
    async function fetchBookFromOpenLibrary(isbn) {
        try {
            const url =
`https://openlibrary.org/api/books?bibkeys=ISBN:${isbn}&format=json&jscmd=data`;
            console.log('📡 OpenLibrary API call:', url);

            const response = await fetch(url);
            const data = await response.json();
            console.log('📡 OpenLibrary Response:', data);

            const bookKey = `ISBN:${isbn}`;
            if (data[bookKey]) {
                const book = data[bookKey];
                const result = {
                    isbn: isbn,
                    title: book.title || 'Unknown Title',
                    authors: book.authors?.map(a => a.name) || ['Unknown Author'],
                    pageCount: book.number_of_pages || 0,
                    coverUrl: book.cover?.medium || '',
                    publishedDate: book.publish_date || 'Unknown'
                };

                console.log('📚 OpenLibrary Result:', result);
                return result;
            }
        } catch (error) {
            debugLog('Open Library API error: ' + error.message);
        }
        return null;
    }

    function displayBookPreview(bookData) {
        const scanInfoHtml = bookData.scanInfo ? `
            <div class="scanned-info">
                📊 ${bookData.scanInfo.source === 'manual' ? 'Manual entry' : 'Scanned'}:
                ${bookData.scanInfo.originalBarcode ? `${bookData.scanInfo.originalBarcode} → `
: ''}
                ISBN: ${bookData.scanInfo.cleanedISBN}
                ${bookData.scanInfo.foundIn ? ` • Found in: ${bookData.scanInfo.foundIn}` : ''}
            </div>
        ` : '';

        document.getElementById('scan-result').innerHTML = `
            <div class="book-card">
```

```
                ${scanInfoHtml}
                <div class="book-info">
                    <img src="${bookData.coverUrl || ''}" alt="${bookData.title}" class="book-cover">
                    <div class="book-details">
                        <h3>${bookData.title}</h3>
                        <p><strong>Author:</strong> ${bookData.authors.join(', ')}</p>
                        <p><strong>Pages:</strong> ${bookData.pageCount || 'Unknown'}</p>
                        <p><strong>Published:</strong> ${bookData.publishedDate}</p>
                    </div>
                </div>
                <div class="book-actions">
                    <button class="btn btn-primary" id="add-book-btn">📚 Add to Library</button>
                    <button class="btn btn-warning" id="continue-scan-btn">🔄 Scan
Another</button>
                    <button class="btn btn-danger" id="wrong-book-btn">❌ Wrong Book</button>
                </div>
            </div>
        `;

        document.getElementById('add-book-btn').addEventListener('click', () =>
addBookToLibrary(bookData));
        document.getElementById('continue-scan-btn').addEventListener('click',
continueScanningAfterPreview);
        document.getElementById('wrong-book-btn').addEventListener('click',
handleWrongBook);
    }

    function handleWrongBook() {
        document.getElementById('scan-result').style.display = 'none';

        if (scanningActive === false && currentStream) {
            scanningActive = true;
            scanForBarcode();
        }

        const manualInput = document.getElementById('manual-isbn');
        if (manualInput) {
            manualInput.value = '';
        }

        showError('Book lookup cancelled. Try scanning a different barcode or check the ISBN
manually.');
    }
```

```
function addBookToLibrary(bookData) {
    if (books.find(book => book.isbn === bookData.isbn)) {
        showError('This book is already in your library!');
        return;
    }

    const newBook = {
        ...bookData,
        id: Date.now(),
        currentPage: 0,
        status: 'reading',
        notes: '',
        recommendation: null,
        dateAdded: new Date().toISOString()
    };

    delete newBook.scanInfo;

    books.push(newBook);
    saveBooks();
    stopScanning();

    const manualInput = document.getElementById('manual-isbn');
    if (manualInput) {
        manualInput.value = '';
    }

    switchTab('library');
    showSuccess('Book added to your library!');
}

function continueScanningAfterPreview() {
    document.getElementById('scan-result').style.display = 'none';

    const manualInput = document.getElementById('manual-isbn');
    if (manualInput) {
        manualInput.value = '';
    }

    scanningActive = true;
    scanForBarcode();
}

function renderLibrary() {
```

```
        const libraryContent = document.getElementById('library-content');

        if (books.length === 0) {
            libraryContent.innerHTML = `
                <div class="empty-state">
                    <h3>Your library is empty</h3>
                    <p>Start by scanning your first book!</p>
                </div>
            `;
            return;
        }

        const booksHtml = books.map(book => {
            const progress = book.pageCount > 0 ? (book.currentPage / book.pageCount) * 100 :
0;

            return `
                <div class="book-card" data-book-id="${book.id}">
                    <div class="book-info">
                        <img src="${book.coverUrl || ''}" alt="${book.title}" class="book-cover">
                        <div class="book-details">
                            <h3>${book.title}</h3>
                            <p><strong>Author:</strong> ${book.authors.join(', ')}</p>
                            ${book.pageCount > 0 ? `<p><strong>Progress:</strong>
${book.currentPage} / ${book.pageCount} pages</p>` : ''}
                        </div>
                    </div>
                    ${book.pageCount > 0 ? `<div class="progress-bar"><div class="progress-fill"
style="width: ${progress}%"></div></div>` : ''}
                    <div class="status-badges">
                        <span class="badge badge-${book.status}">
                            ${book.status === 'reading' ? '📖 Reading' : book.status === 'finished' ? '✅
Finished' : '⏸️ DNF'}
                        </span>
                        ${book.recommendation ? `<span class="badge
badge-${book.recommendation}">${book.recommendation === 'recommend' ? '👍
Recommend' : '👎 Not Recommend'}</span>` : ''}
                    </div>
                    <div class="page-input">
                        <label>Current Page:</label>
                        <input type="number" class="page-input-field" value="${book.currentPage}"
min="0" max="${book.pageCount || 9999}">
                        ${book.pageCount > 0 ? `<span>/ ${book.pageCount}</span>` : ''}
                    </div>
```

```
                    <textarea class="notes-area" placeholder="Add your reading
notes...">${book.notes}</textarea>
                    <div class="book-actions">
                        <button class="btn status-btn ${book.status === 'reading' ? 'btn-primary' :
'btn-warning'}" data-status="reading">📖 Reading</button>
                        <button class="btn status-btn ${book.status === 'finished' ? 'btn-success' :
'btn-warning'}" data-status="finished">✅ Finished</button>
                        <button class="btn status-btn ${book.status === 'dnf' ? 'btn-warning' :
'btn-danger'}" data-status="dnf">⏸️ DNF</button>
                        <button class="btn rec-btn ${book.recommendation === 'recommend' ?
'btn-success' : 'btn-warning'}" data-recommendation="recommend">👍 Recommend</button>
                        <button class="btn rec-btn ${book.recommendation === 'TBR' ? 'btn-danger' :
'btn-warning'}" data-recommendation="not-recommend">📌 TBR</button>
                        <button class="btn btn-danger remove-btn">🗑️ Remove</button>
                    </div>
                </div>
            `;
        }).join('');

        libraryContent.innerHTML = booksHtml;

        // Add event listeners
        document.querySelectorAll('.book-card').forEach(card => {
            const bookId = parseInt(card.getAttribute('data-book-id'));

            card.querySelector('.page-input-field').addEventListener('change', (e) =>
updateBookPage(bookId, e.target.value));
            card.querySelector('.notes-area').addEventListener('change', (e) =>
updateBookNotes(bookId, e.target.value));

            card.querySelectorAll('.status-btn').forEach(btn => {
                btn.addEventListener('click', (e) => updateBookStatus(bookId,
e.target.getAttribute('data-status')));
            });

            card.querySelectorAll('.rec-btn').forEach(btn => {
                btn.addEventListener('click', (e) => updateBookRecommendation(bookId,
e.target.getAttribute('data-recommendation')));
            });

            card.querySelector('.remove-btn').addEventListener('click', () =>
removeBook(bookId));
        });
    }
```

```
function updateBookPage(bookId, newPage) {
    const book = books.find(b => b.id === bookId);
    if (book) {
        book.currentPage = parseInt(newPage) || 0;
        if (book.pageCount > 0 && book.currentPage >= book.pageCount) {
            book.status = 'finished';
        }
        saveBooks();
        renderLibrary();
    }
}

function updateBookNotes(bookId, notes) {
    const book = books.find(b => b.id === bookId);
    if (book) {
        book.notes = notes;
        saveBooks();
    }
}

function updateBookStatus(bookId, status) {
    const book = books.find(b => b.id === bookId);
    if (book) {
        book.status = status;
        if (status === 'finished' && book.pageCount > 0) {
            book.currentPage = book.pageCount;
        }
        saveBooks();
        renderLibrary();
    }
}

function updateBookRecommendation(bookId, recommendation) {
    const book = books.find(b => b.id === bookId);
    if (book) {
        book.recommendation = book.recommendation === recommendation ? null :
recommendation;
        saveBooks();
        renderLibrary();
    }
}

function removeBook(bookId) {
```

```
        if (confirm('Are you sure you want to remove this book from your library?')) {
            books = books.filter(b => b.id !== bookId);
            saveBooks();
            renderLibrary();
        }
    }

    function showError(message) {
        const errorDiv = document.createElement('div');
        errorDiv.className = 'error';
        errorDiv.textContent = message;

        const scanArea = document.querySelector('#scan-tab > div');
        scanArea.insertBefore(errorDiv, scanArea.firstChild);

        setTimeout(() => {
            if (errorDiv.parentNode) {
                errorDiv.parentNode.removeChild(errorDiv);
            }
        }, 5000);
    }

    function showSuccess(message) {
        const successDiv = document.createElement('div');
        successDiv.style.cssText = 'background: #333; color: #cc5500; padding: 15px;
border-radius: 8px; margin: 20px; text-align: center; font-weight: bold; border: 1px solid
#cc5500;';
        successDiv.textContent = message;

        document.querySelector('.header').after(successDiv);

        setTimeout(() => {
            if (successDiv.parentNode) {
                successDiv.parentNode.removeChild(successDiv);
            }
        }, 3000);
    }

    // Handle page visibility changes
    document.addEventListener('visibilitychange', function() {
        if (document.hidden && scanningActive) {
            stopScanning();
        }
    });
```

```
        // Clear any browser cache or localStorage contamination on startup
        window.addEventListener('load', function() {
            // Clear any potential contaminated variables
            window.cachedBarcodeResult = null;
            window.lastFrameData = null;
            window.detectedBarcodes = [];

            console.log('🚀 App fully loaded and cleaned');
        });
    </script>
</body>
</html>
```