Cloudflare NFC Gospel Project - Developer Documentation

This document provides a comprehensive guide for developing the Cloudflare-powered NFC Gospel web application. It covers architecture, feature implementation, and deployment strategies, aiming for high stability, performance, and cost-efficiency by leveraging Cloudflare's serverless and edge computing capabilities.

1. Project Overview

The NFC Gospel project is a web application triggered by NFC tags, designed to deliver a dynamic and personalized experience. Key functionalities include:

- Automatic language detection based on user IP.
- Dynamic language and Bible version switching.
- Anonymous user name submission for tracking "gospel reach."
- Random Bible verse display with refresh functionality.
- Real-time global view counter and map visualization.
- Randomized background photos or videos.
- Integration with Google AdSense for monetization.

The entire application stack is built on Cloudflare, minimizing external dependencies and maximizing edge performance.

2. Core Cloudflare Components Utilized

- **Cloudflare Workers:** The primary serverless compute platform for handling all dynamic logic, API endpoints, data fetching, language detection, verse selection, and counter updates.
- **Cloudflare KV (Key-Value Store):** Used for storing small, structured data such as Bible verse texts, language and version configurations, and anonymized user name submissions.
- **Cloudflare R2 (Object Storage):** Employed for storing larger media files, specifically the collection of background images and videos.
- **Cloudflare Durable Objects:** Provides a globally consistent, real-time counter for total page views and verse refreshes, ensuring atomic increments for accurate statistics.
- **Cloudflare Pages / Workers Sites:** Serves as the hosting platform for all static frontend assets (HTML, CSS, JavaScript).
- **Cloudflare DNS:** Manages domain name resolution for the application.
- **Cloudflare CDN:** Automatically caches and delivers both static and dynamic content from Cloudflare's global edge network, ensuring low latency and high

availability.
- **Cloudflare Turnstile:** A privacy-preserving CAPTCHA alternative used to verify human users and prevent bot abuse on specific interactive actions (e.g., name submission).

3. Project Structure

The project will be organized into two main parts: the frontend (static assets) and the backend (Cloudflare Worker).

```
project-root/
├── public/               # Frontend static assets (hosted by Cloudflare Pages/Workers Sites)
│   ├── index.html        # Main HTML structure of the web page
│   ├── css/
│   │   └── style.css     # Core CSS for styling the application
│   └── js/
│       └── app.js        # Main frontend JavaScript for interactivity, API calls, and UI updates
├── worker/               # Cloudflare Worker source code and configuration
│   ├── src/
│   │   ├── index.js      # Worker entry point: main logic, routing, and API handlers
│   │   ├── utils.js      # Utility functions (e.g., UUID generation, random selection helpers)
│   │   └── data/         # Directory for static data bundled with the Worker
│   │       ├── languages.json # Configuration for languages and their supported Bible versions
│   │       ├── bible_index.json # Index mapping for random Bible verse selection
│   │       └── background_media_index.json # Index for random background media selection
│   └── wrangler.toml     # Cloudflare Worker configuration file (bindings, environment variables)
├── .env.example          # Example file for local environment variables (e.g., Turnstile secret)
└── README.md             # Project README file
```

4. Detailed Feature Requirements & Implementation

4.1. Language Detection & Switching

Objective: Automatically detect the user's country to set a default language, and provide a user-friendly interface for language and Bible version switching.

Implementation Details:

- **Default Language Detection (Cloudflare Worker):**
  - On every incoming HTTP request, the Worker will access the request.headers.get('CF-IPCountry') header. This header is automatically populated by Cloudflare with the two-letter ISO country code of the client's IP address.
  - The Worker will use a predefined mapping (loaded from worker/src/data/languages.json) to determine the default language for that country (e.g., US, GB, AU, CA, NZ map to English (EN); FR maps to French (FR), etc.).
  - The initial HTML response rendered by the Worker will embed the content in this detected default language, including the default Bible version for that language.
- **Language Configuration (Worker Bundled Asset):**
  - A JSON file (worker/src/data/languages.json) will define all supported languages, their display names, their default Bible version, and a list of available Bible versions for each language. This data will be bundled directly with the Worker for fast access.

```
{
 "EN": {
  "name": "English",
  "defaultVersion": "KJV",
  "versions": {
   "KJV": "King James Version",
   "NIV": "New International Version",
   "ESV": "English Standard Version"
  }
 },
 "FR": {
  "name": "Français",
  "defaultVersion": "LSG",
  "versions": {
   "LSG": "Louis Segond 1910",
   "BDS": "La Bible du Semeur"
  }
 },
 "IT": {
```

```
    "name": "Italiano",
    "defaultVersion": "CEI",
    "versions": {
      "CEI": "Conferenza Episcopale Italiana",
      "NR94": "Nuova Riveduta 1994"
    }
  },
  "ES": {
    "name": "Español",
    "defaultVersion": "RVR60",
    "versions": {
      "RVR60": "Reina-Valera 1960",
      "NVI": "Nueva Versión Internacional"
    }
  },
  "DE": {
    "name": "Deutsch",
    "defaultVersion": "LUTH2017",
    "versions": {
      "LUTH2017": "Lutherbibel 2017",
      "ELB": "Elberfelder Bibel"
    }
  }
}
```

- **Language Switcher (Frontend - public/js/app.js):**
  - The UI will feature five prominent buttons (EN, FR, IT, ES, DE) in the top-right corner of the page.
  - When a user clicks a language button:
    1. The frontend JavaScript will update a URL query parameter (e.g., ?lang=FR) to reflect the selected language.
    2. It will then make an API call to the Worker (e.g., GET /api/config?lang=FR) to fetch the specific configuration for the newly selected language.
    3. Upon receiving the response, the frontend will dynamically update the "Bible Version" dropdown or button group to display **only** the versions available for the selected language.
    4. The defaultVersion for the newly selected language will be automatically chosen in the version selector, and a request to display a random verse from this default version will be initiated.
- **API Endpoint for Language/Version Config (Worker - worker/src/index.js):**
  - GET /api/config?lang=<LANG_CODE>:
    - **Request:** Expects a lang query parameter (e.g., EN, FR).

    - **Processing:** The Worker will load the languages.json data. It will then retrieve the configuration object corresponding to the provided <LANG_CODE>.
    - **Response:** Returns a JSON object containing the name, defaultVersion, and versions map for the requested language. If the language code is invalid, an appropriate error response or a default language config will be returned.

4.2. User Identification & Anonymous Name Submission

Objective: Allow users to voluntarily submit their name, associating it with an anonymous identifier to track "gospel reach" without compromising privacy. IMEI number acquisition is strictly prohibited.

Implementation Details:

- **IMEI Number: Crucially, direct access to the IMEI number is technically impossible and strictly forbidden in a web browser environment due to privacy and security restrictions. Any attempts to obtain it will fail and are against best practices.**
- **Anonymous User ID (Frontend - public/js/app.js):**
    - On every page load, app.js will attempt to retrieve a unique identifier from localStorage using a key like gospel_user_id.
    - If no ID is found, a **UUID v4** (Universally Unique Identifier, version 4) will be generated by the JavaScript and stored in localStorage under gospel_user_id. This ID is anonymous and does not contain any personal information.
    - This ID will persist across browser sessions on the same device unless the user explicitly clears their browser's local storage.
- **Name Submission Form (Frontend - public/index.html & public/js/app.js):**
    - A simple HTML form with an input field for user_name will be present on the page.
    - This form will also include a **Cloudflare Turnstile widget**.
    - When the user submits the form:
        1. Frontend JavaScript will collect the user_name from the input field.
        2. It will retrieve the gospel_user_id from localStorage.
        3. It will obtain the cf-turnstile-response token generated by the Turnstile widget.
        4. These three pieces of data (user_name, gospel_user_id, cf-turnstile-response) will be sent as a POST request to the Worker's /api/submit-name endpoint using FormData.
- **Name Submission Processing (Cloudflare Worker - worker/src/index.js):**

    - POST /api/submit-name:
        - **Request:** Expects user_name, user_id (the UUID), and cf-turnstile-response in the request body (FormData).
        - **Turnstile Validation:** The Worker will first validate the cf-turnstile-response token by making a POST request to Cloudflare's Turnstile verification API (https://challenges.cloudflare.com/turnstile/v0/siteverify). The TURNSTILE_SECRET_KEY (stored as a Worker Environment Secret) will be used for this validation. If the token is invalid, the request will be rejected with a 403 Forbidden status.
        - **Data Storage (Cloudflare KV - USER_SUBMISSIONS_KV binding):** If Turnstile validation passes, the Worker will store the submitted data in a dedicated Cloudflare KV namespace.
            - **Key:** user_submission_<UUID> (e.g., user_submission_a1b2c3d4-e5f6-7890-1234-567890abcdef).
            - **Value:** A JSON string containing the submitted name and a timestamp: {"name": "Submitted Name", "timestamp": "ISO_DATE_STRING"}.
            - This ensures that each anonymous ID (representing a browser/device) can submit a name once, fulfilling the "who this gospel was shared with" requirement anonymously.
        - **Response:** Returns a success message or an error if data is missing/invalid.

4.3. Bible Verse Version Switching & Random Refresh

Objective: Provide users with the ability to switch between different Bible versions for the selected language and refresh the displayed verse randomly.

Implementation Details:

- **Bible Verse Storage (Cloudflare KV - BIBLE_TEXT_KV binding):**
    - All 365 pre-selected Bible verses, translated into all 5 languages and their respective 11 versions, will be stored as individual key-value pairs in a dedicated Cloudflare KV namespace.
    - **Key Format:** bible_text_<LANG_CODE>_<VERSION_CODE>_<VERSE_INDEX_001_to_365> (e.g., bible_text_EN_KJV_001, bible_text_FR_LSG_123).
    - **Value:** The plain text content of the Bible verse.
    - This approach is suitable as the total data volume (~1MB) fits comfortably within KV's limits and offers fast random access.
- **Verse Index (Worker Bundled Asset - worker/src/data/bible_index.json):**

    - A JSON array containing sequential identifiers for the 365 verses (e.g., ["001", "002", ..., "365"]) will be bundled with the Worker. This allows for efficient random selection without additional KV lookups for the index itself.
- **API Endpoint for Random Verse (Cloudflare Worker - worker/src/index.js):**
    - GET /api/verse?lang=<LANG_CODE>&version=<VERSION_CODE>:
        - **Request:** Requires lang and version query parameters.
        - **Validation:** The Worker will first validate that the requested <VERSION_CODE> is indeed a valid version for the given <LANG_CODE> by checking the languages.json configuration. If not valid, it will return an error or default to the defaultVersion for that language.
        - **Random Selection:** The Worker will load the bible_index.json data. It will then generate a random integer between 0 and 364 to select a VERSE_INDEX from this array.
        - **KV Fetch:** A KV key will be constructed using the provided LANG_CODE, validated VERSION_CODE, and the randomly selected VERSE_INDEX. The Worker will then fetch the corresponding verse text from the BIBLE_TEXT_KV namespace.
        - **Response:** Returns a JSON object containing the verse text (e.g., { "verse": "For God so loved the world..." }).
- **Frontend Logic (public/js/app.js):**
    - A prominent "Refresh" button will be displayed next to the Bible verse.
    - Clicking this button will trigger a fetch request to the /api/verse endpoint, passing the currently selected language and version.
    - The fetched verse text will then dynamically update the content of the Bible verse display area on the page.

4.4. Real-time Global View Counter & Map

Objective: Display a real-time global counter of "gospel activations" (page loads/verse refreshes) and visualize these activations on a world map.

Implementation Details:

- **Global Counter (Cloudflare Durable Object - GLOBAL_COUNTER_DO binding):**
    - A single instance of a Cloudflare Durable Object (e.g., named GlobalCounter) will be used to maintain the total count of "gospel activations."
    - Each successful call to the /api/verse endpoint (representing a page load or verse refresh) will trigger an **atomic increment** operation on this Durable Object. Durable Objects ensure strong consistency and atomic operations for shared state across the globe.

- **Durable Object Class (worker/src/index.js):**

```
// Example Durable Object class (within worker/src/index.js or a separate file
imported)
export class GlobalCounter {
  constructor(state, env) {
    this.state = state;
    this.env = env;
    this.storage = state.storage; // Durable Object's built-in storage
  }

  async fetch(request) {
    let url = new URL(request.url);
    // Retrieve current counts, defaulting to 0 or empty object if not found
    let currentTotalCount = (await this.storage.get('total_count')) || 0;
    let currentCountryCounts = (await this.storage.get('country_counts')) || {};

    if (url.pathname === '/increment') {
      const country = url.searchParams.get('country'); // Get country from query
param
      currentTotalCount++; // Increment global count
      if (country) {
        // Increment country-specific count
        currentCountryCounts[country] = (currentCountryCounts[country] || 0) +
1;
      }
      // Atomically update storage
      await this.storage.put('total_count', currentTotalCount);
      await this.storage.put('country_counts', currentCountryCounts);
      return new Response(JSON.stringify({ total: currentTotalCount, countries:
currentCountryCounts }), { headers: { 'Content-Type': 'application/json' } });
    } else if (url.pathname === '/get') {
      // Return current counts
      return new Response(JSON.stringify({ total: currentTotalCount, countries:
currentCountryCounts }), { headers: { 'Content-Type': 'application/json' } });
    }
    return new Response('Not found', { status: 404 });
  }
}
```

    - **Worker Interaction with DO (worker/src/index.js):**
        - When the Worker handles a GET /api/verse request, after fetching the Bible verse, it will interact with the Durable Object.
        - It will get the Durable Object instance (e.g., env.GLOBAL_COUNTER_DO.idFromName('main_counter').get(request.url)) and make an RPC call to its /increment method, passing the CF-IPCountry header value as a query parameter.
- **Country-Specific Counts (Durable Object Storage):**
    - Within the same GlobalCounter Durable Object, a map (e.g., country_counts) will store the activation count for each country (derived from CF-IPCountry).
    - When the Durable Object's /increment method is called, it will increment both the total_count and the specific country's count.
- **API Endpoint for Stats (Cloudflare Worker - worker/src/index.js):**
    - GET /api/stats:
        - **Request:** No parameters needed.
        - **Processing:** The Worker will fetch the GlobalCounter Durable Object instance and make an RPC call to its /get method.
        - **Response:** Returns a JSON object containing the total_count and the country_counts map (e.g., { "total": 12345, "countries": {"US": 100, "FR": 50, ...}}).
- **Real-time Update & Map Visualization (Frontend - public/js/app.js):**
    - **Polling:** The frontend JavaScript will implement a polling mechanism (e.g., using setInterval to call /api/stats every 5-10 seconds) to fetch the latest counts.
    - **Map Library:** An interactive JavaScript mapping library (e.g., Leaflet.js or Mapbox GL JS) will be used to render a world map.
    - **Visualization:** The country_counts data received from /api/stats will be used to dynamically update the map. This could involve:
        - Shading countries based on their activation count (e.g., darker shade for more activations).
        - Placing markers or displaying numbers on countries.
        - Implementing a subtle "lighting up" animation or visual effect on a country when its count increments, providing real-time feedback.

4.5. Random Background Photo/Video

Objective: Dynamically display a random background photo or video that covers the entire page.

Implementation Details:

- **Resource Storage (Cloudflare R2 - BACKGROUND_MEDIA_R2 binding):**
  - All background images (JPG, PNG, WebP) and videos (MP4, WebM) will be stored in a dedicated Cloudflare R2 bucket.
  - The R2 bucket should be configured for public access. If a custom domain is bound to the R2 bucket, use that URL for direct access. Otherwise, Cloudflare Pages can proxy R2 assets.
- **Background Media Index (Worker Bundled Asset - worker/src/data/background_media_index.json):**
  - A JSON array listing the URLs or R2 object keys of all available background media will be bundled with the Worker.
  - **Format:** Each entry will be an object { "type": "image" | "video", "url": "https://your-r2-domain.com/path/to/media.ext" }.
- **API Endpoint for Random Background (Cloudflare Worker - worker/src/index.js):**
  - GET /api/random-background:
    - **Request:** No parameters needed.
    - **Processing:** The Worker will load the background_media_index.json array. It will then generate a random index to select one media entry from the array.
    - **Response:** Returns a JSON object containing the type (e.g., "image" or "video") and the url of the selected media (e.g., { "type": "image", "url": "https://your-r2-domain.com/bg/nature1.jpg" }).
- **Frontend Logic (public/js/app.js & public/css/style.css):**
  - **HTML Structure (public/index.html):** A container element (e.g.,) will be used to hold the background media.
  - **CSS Styling (public/css/style.css):**

```
#background-container {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  z-index: -1; /* Ensure it's behind content */
  overflow: hidden;
}
#background-container img,
#background-container video {
  width: 100%;
```

```
    height: 100%;
    object-fit: cover; /* Cover the container, cropping if necessary */
    display: block;
    /* Optional: Add transition for smoother changes */
    transition: opacity 0.5s ease-in-out;
}
```

- **JavaScript (public/js/app.js):**
  1. On page load (and optionally when the verse refresh button is clicked, or via a dedicated background refresh button), a fetch request will be made to /api/random-background.
  2. Upon receiving the JSON response, JavaScript will dynamically create either an <img> or <video> element based on the type property.
  3. The src attribute of the created element will be set to the url from the response.
  4. For <video> elements, autoplay, loop, muted, and playsinline attributes are crucial for automatic playback on most mobile devices.
  5. The newly created media element will be appended to the #background-container, replacing any previously loaded background.

4.6. Google AdSense Integration

Objective: Display Google AdSense advertisements at the bottom of the page for monetization.

Implementation Details:

- **Frontend-Only Integration (public/index.html):**
  - Google AdSense is a client-side advertising platform. Its integration does not involve the Cloudflare Worker or other backend components beyond serving the static HTML.
  - The AdSense provided JavaScript code snippet and ad unit code will be directly embedded into the public/index.html file, typically within a dedicated div element at the bottom of the <body>.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <script async
src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js?client
=ca-pub-YOUR_ADSENSE_PUBLISHER_ID" crossorigin="anonymous"></script>
</head>
<body>
```

```
    <div id="app-content">
    </div>

    <div class="ad-container" style="text-align: center; margin-top: 20px;">
        <ins class="adsbygoogle"
            style="display:block"
            data-ad-client="ca-pub-YOUR_ADSENSE_PUBLISHER_ID"
            data-ad-slot="YOUR_AD_UNIT_SLOT_ID"
            data-ad-format="auto"
            data-full-width-responsive="true"></ins>
        <script>
            (adsbygoogle = window.adsbygoogle || []).push({});
        </script>
    </div>

    </body>
</html>
```

    - **IMPORTANT:** Replace ca-pub-YOUR_ADSENSE_PUBLISHER_ID and YOUR_AD_UNIT_SLOT_ID with your actual values obtained from your Google AdSense account.
- **Content Security Policy (CSP) Considerations:**
    - If you implement a Content Security Policy (which is recommended for security), you **must** whitelist the necessary Google AdSense domains.
    - Example CSP directives (add to your HTML <meta> tag or Cloudflare Pages/Worker Sites security settings):
      ```
      Content-Security-Policy:
        default-src 'self';
        script-src 'self' https://pagead2.googlesyndication.com
      https://*.doubleclick.net;
        frame-src https://*.doubleclick.net;
        connect-src 'self' https://pagead2.googlesyndication.com;
        img-src 'self' https://*.doubleclick.net https://*.googlesyndication.com;
      ```
    - **Note:** This is a basic example; always consult Google AdSense documentation for the most up-to-date CSP requirements.

5. API Endpoints Summary (Cloudflare Worker)

All API endpoints will be handled by the Cloudflare Worker (worker/src/index.js).

- **GET /**:
    - **Purpose:** Initial page load. Renders the main index.html with dynamic content

(default language, initial random verse, background).
    - **Logic:** Detects default language via CF-IPCountry, fetches initial verse and background, increments global counter.
- **GET /api/config?lang=<LANG_CODE>**:
    - **Purpose:** Retrieve configuration for a specific language, including its supported Bible versions.
    - **Parameters:** lang (e.g., EN, FR).
    - **Response:** JSON object containing language details (name, defaultVersion, versions).
- **POST /api/submit-name**:
    - **Purpose:** Allow anonymous users to submit their name, linked to their anonymous UUID.
    - **Parameters (FormData):** user_name, user_id (UUID), cf-turnstile-response (Turnstile token).
    - **Logic:** Validates Turnstile token, stores user_id and user_name in Cloudflare KV.
- **GET /api/verse?lang=<LANG_CODE>&version=<VERSION_CODE>**:
    - **Purpose:** Fetch a random Bible verse for the specified language and version.
    - **Parameters:** lang, version.
    - **Logic:** Validates language/version, randomly selects a verse index, fetches verse text from Cloudflare KV.
    - **Side Effect:** Increments the global and country-specific counters in the Durable Object.
    - **Response:** JSON object with the verse text.
- **GET /api/stats**:
    - **Purpose:** Retrieve real-time global and country-specific view counts.
    - **Parameters:** None.
    - **Logic:** Queries the Global Counter Durable Object.
    - **Response:** JSON object with total count and countries map.
- **GET /api/random-background**:
    - **Purpose:** Fetch a URL for a random background image or video.
    - **Parameters:** None.
    - **Logic:** Randomly selects an entry from the bundled background media index.
    - **Response:** JSON object with type (image/video) and url.

6. Environment Variables & Security

Sensitive information (like API keys) must be stored securely and never hardcoded in the source code.

- wrangler.toml (for local development and deployment configuration):
  This file defines the Worker's bindings to Cloudflare services (KV, R2, Durable Objects) and declares environment variables.
  name = "nfc-gospel-worker"
  main = "worker/src/index.js" # Path to your Worker's entry file
  compatibility_date = "2025-05-22" # Use a recent date for compatibility

  # KV Namespace Bindings
  [[kv_namespaces]]
  binding = "BIBLE_TEXT_KV"   # Name used in Worker code (e.g., `env.BIBLE_TEXT_KV`)
  id = "<YOUR_BIBLE_TEXT_KV_NAMESPACE_ID>" # Replace with your actual KV Namespace ID

  [[kv_namespaces]]
  binding = "USER_SUBMISSIONS_KV"   # Name used in Worker code
  id = "<YOUR_USER_SUBMISSIONS_KV_NAMESPACE_ID>" # Replace with your actual KV Namespace ID

  # R2 Bucket Binding
  [[r2_buckets]]
  binding = "BACKGROUND_MEDIA_R2"   # Name used in Worker code (e.g., `env.BACKGROUND_MEDIA_R2`)
  bucket_name = "<YOUR_R2_BUCKET_NAME>" # Replace with your actual R2 Bucket Name

  # Durable Object Binding
  [[durable_objects.bindings]]
  name = "GLOBAL_COUNTER_DO"   # Name used in Worker code (e.g., `env.GLOBAL_COUNTER_DO`)
  class_name = "GlobalCounter"   # Name of the Durable Object class in your Worker code
  script_name = "nfc-gospel-worker" # If DO class is defined in this same Worker script

  # Environment Variables (for non-sensitive data, or placeholders for secrets)
  [vars]
  # PUBLIC_R2_BASE_URL = "https://<YOUR_R2_PUBLIC_DOMAIN>" # Uncomment if you bind a custom domain to R2

```
# Secrets (for sensitive data, managed by `wrangler secret put`)
[secrets]
# TURNSTILE_SECRET_KEY = "<YOUR_TURNSTILE_SECRET_KEY>" # This will be set via `wrangler secret put`
```

- **Cloudflare Worker Secrets:**
  - Sensitive variables like TURNSTILE_SECRET_KEY should be set as secrets in the Cloudflare Dashboard or via the wrangler secret put CLI command. They are not stored directly in wrangler.toml or the source code.
  - Example CLI command: wrangler secret put TURNSTILE_SECRET_KEY --name <YOUR_WORKER_NAME> (then paste the key when prompted).

7. Deployment Workflow

This section outlines the steps to deploy your application on Cloudflare.

1. **Cloudflare Account Setup:**
   - Ensure you have a Cloudflare account.
   - Create necessary Cloudflare resources:
     - Two KV Namespaces (e.g., bible_text_kv, user_submissions_kv). Note their IDs.
     - One R2 Bucket (e.g., background_media_r2). Note its name.
     - A Durable Object namespace (this is created automatically when you deploy a Worker with a Durable Object binding, but ensure the class_name and name in wrangler.toml match your Worker code).
     - A Turnstile Sitekey and Secret Key from the Cloudflare Dashboard.
2. **Data Preparation & Upload:**
   - **Bible Verses:** Format your 365 Bible verses for all languages/versions. Use wrangler kv:bulk put --binding=BIBLE_TEXT_KV --file=path/to/verses.json (or a custom script) to upload them to the BIBLE_TEXT_KV namespace. The verses.json should be an array of objects like {"key": "bible_text_EN_KJV_001", "value": "..."}.
   - **Background Media:** Optimize your background images and videos. Upload them to your R2 bucket using wrangler r2 object put <FILE_PATH> --bucket <YOUR_R2_BUCKET_NAME> or through the Cloudflare Dashboard. Ensure public access is configured or a custom domain is set up for R2.
3. **Frontend Deployment (Cloudflare Pages):**
   - Commit your public/ directory to a Git repository (e.g., GitHub, GitLab).
   - In the Cloudflare Dashboard, navigate to "Workers & Pages" -> "Create

application" -> "Pages" -> "Connect to Git."
    - Select your repository and configure the build settings (usually default for plain HTML/CSS/JS).
    - Deploy the Pages project. This will host your static index.html, style.css, and app.js.
4. **Worker Deployment (Cloudflare Workers):**
    - Ensure your worker/src/index.js (or .ts) is complete and wrangler.toml is correctly configured with all bindings and the class_name for your Durable Object.
    - Set your TURNSTILE_SECRET_KEY as a secret for your Worker: wrangler secret put TURNSTILE_SECRET_KEY --name <YOUR_WORKER_NAME>.
    - Deploy the Worker using the wrangler CLI: wrangler deploy. This will deploy your Worker code and create/update the Durable Object namespace.
5. **DNS Configuration & Routing:**
    - Ensure your domain is managed by Cloudflare (DNS records proxied through Cloudflare).
    - In the Cloudflare Dashboard, navigate to "Workers & Pages" -> "Your Worker" -> "Triggers" -> "Custom Domains." Add your custom domain (e.g., your-domain.com) and configure the route (e.g., your-domain.com/*) to point to your Worker. This ensures all traffic for your domain goes through your Worker first.
    - If using Cloudflare Pages for the frontend, ensure its custom domain is also set up and points to Pages. The Worker will then handle the root path (/) and API paths, while Pages serves other static assets.

8. Development Notes & Best Practices

- **Local Development:** Use wrangler dev to run your Worker locally. This allows you to test API endpoints and Worker logic without deploying to Cloudflare's edge.
- **Error Handling:** Implement robust try...catch blocks in your Worker code to gracefully manage API call failures (e.g., KV/R2 read errors, Turnstile verification failures) and return appropriate HTTP status codes and error messages to the frontend.
- **Frontend Error Display:** The frontend (app.js) should be designed to gracefully handle and display errors from the Worker (e.g., a message box instead of alert()).
- **Caching Strategy:**
    - Cloudflare's CDN automatically caches static assets (HTML, CSS, JS, R2 media).
    - For Worker responses, consider setting Cache-Control headers (e.g., Cache-Control: public, max-age=3600) on API responses that can be cached

(e.g., /api/config, /api/random-background) to further reduce Worker executions and latency.
- **Observability:**
  - Utilize Cloudflare's built-in analytics and logging for Workers (Workers Analytics, Logs tab in Dashboard) to monitor execution times, errors, and traffic patterns.
  - Durable Objects also provide their own specific metrics.
- **Code Quality:**
  - Use **TypeScript** for Worker development (index.ts, utils.ts) to leverage static type checking, improve code readability, and reduce runtime errors.
  - Follow consistent coding standards and add comments for complex logic.
- **Performance Optimization:**
  - Keep Worker code lean to minimize CPU time and cold starts.
  - Optimize image and video assets for web (compression, appropriate formats) to reduce R2 bandwidth and improve load times.
- **Privacy Compliance:** Ensure your website has a clear privacy policy that explains what data is collected (anonymous UUIDs, voluntarily submitted names), how it's used, and how users can control it (e.g., by clearing local storage). This is crucial for GDPR, CCPA, and other privacy regulations.
- **No alert() or confirm():** Avoid using browser native alert() or confirm() calls in your JavaScript, as they can be blocked or provide a poor user experience in an iframe environment. Implement custom modal dialogs instead.

9. Privacy Collection Document (GDPR Compliant Guidelines)

This section outlines the key elements for a Privacy Policy document, designed to comply with GDPR standards, specifically addressing data collection and cookie consent relevant to the NFC Gospel project.

Disclaimer: This is a guideline for content. **Always consult with legal counsel** to ensure your Privacy Policy fully complies with all applicable laws and regulations for your specific jurisdiction and use case.

Privacy Policy for [Your Website Name]

Effective Date: [Current Date, e.g., May 22, 2025]

This Privacy Policy explains how [Your Website Name] ("we," "us," or "our") collects, uses, and protects information when you visit our website via NFC tags or direct access. We are committed to protecting your privacy and handling your data transparently and in compliance with the General Data Protection Regulation (GDPR)

and other relevant privacy laws.

1. What Information We Collect

We collect the following types of information:

- **Anonymous Usage Data:**
  - **Country of Origin (IP-derived):** When you access our website, Cloudflare, our content delivery network, automatically provides us with the two-letter ISO country code derived from your IP address (CF-IPCountry header). This is used solely to determine your default language preference and for anonymous, aggregated geographical statistics on "gospel activations." Your full IP address is not stored by us.
  - **Anonymous User ID (UUID):** For functional purposes, our website generates and stores a unique, anonymous identifier (UUID v4) in your browser's local storage (not a traditional cookie). This ID helps us remember your language/version preferences and allows for anonymous, aggregated tracking of "gospel activations" from your specific browser/device. This ID does not identify you personally.
  - **Interaction Data:** We track anonymous counts of page loads, verse refreshes, and language/version switches to understand website engagement. These counts are aggregated globally and by country.
- **Voluntarily Provided Information:**
  - **Your Name:** If you choose to use our "Submit Your Name" feature, you voluntarily provide your name. This name is associated with your anonymous User ID (UUID) and is used solely for the purpose of anonymously acknowledging "who this gospel was shared with" in our aggregated statistics. We do not attempt to link this name back to your personal identity.
- **Third-Party Information (Google AdSense):**
  - Our website uses Google AdSense to display advertisements. Google AdSense may collect and use data about your visits to this and other websites to provide advertisements about goods and services of interest to you. This may involve the use of cookies and similar technologies by Google and its partners. Please refer to Google's Privacy Policy for more information on how they handle data.

2. How We Use Your Information

We use the collected information for the following purposes:

- **Website Functionality:** To provide the core features of our website, including:
  - Automatically setting your default language based on your country.

    - Remembering your language and Bible version preferences.
    - Displaying random Bible verses.
    - Enabling the "Submit Your Name" feature for anonymous acknowledgment.
- **Analytics and Statistics:** To understand website usage patterns, measure the global reach of the "gospel," and improve our services. All analytical data is aggregated and anonymized.
- **Security:** To protect our website from malicious activity, such as DDoS attacks, using Cloudflare's security services, including Turnstile for bot verification.
- **Advertising:** To display relevant advertisements through Google AdSense, which helps support the operation of this website.

3. Legal Basis for Processing (GDPR)

We process your data based on the following legal grounds:

- **Legitimate Interests:** We process anonymous usage data (country of origin, anonymous user ID, interaction data) to operate, maintain, and improve our website, and to understand its global reach. This processing is necessary for our legitimate interests in providing a functional and engaging service, and these interests are balanced against your data protection rights.
- **Consent:** When you voluntarily submit your name, you provide explicit consent for us to process this information for the stated purpose of anonymous acknowledgment. You can withdraw this consent at any time.
- **Contractual Necessity:** For certain website functionalities that you explicitly request (e.g., interacting with the name submission form), processing your anonymous ID is necessary to fulfill that request.

4. Data Sharing and Disclosure

We do not share, sell, or rent your personal data to third parties.

- **Service Providers:** We use Cloudflare as our content delivery network and serverless platform. Cloudflare processes data on our behalf to provide its services (e.g., routing traffic, DDoS protection, KV/R2 storage, Durable Objects). Cloudflare is committed to GDPR compliance.
- **Google AdSense:** As mentioned, Google AdSense operates independently and may collect data for advertising purposes. We do not control Google's data practices.
- **Legal Requirements:** We may disclose your information if required to do so by law or in response to valid requests by public authorities (e.g., a court order or government agency).

5. Data Retention

- **Anonymous User ID:** Stored in your browser's local storage until you clear your browser data.
- **Voluntarily Submitted Names:** Stored in Cloudflare KV for the duration necessary for our anonymous statistical purposes, typically [e.g., 2 years], after which they may be anonymized further or deleted.
- **Aggregated Statistics:** Global and country-specific counts are retained indefinitely as they are anonymized and do not identify individuals.

6. Your Data Protection Rights (GDPR)

Under GDPR, you have the following rights regarding your data:

- **Right to Access:** You have the right to request copies of your personal data we hold (if any identifiable data is collected, which is minimal in this project).
- **Right to Rectification:** You have the right to request that we correct any information you believe is inaccurate or complete information you believe is incomplete.
- **Right to Erasure ("Right to be Forgotten"):** You have the right to request that we erase your personal data under certain conditions. For the anonymous User ID, you can do this by clearing your browser's local storage. For voluntarily submitted names, you can contact us to request deletion.
- **Right to Restrict Processing:** You have the right to request that we restrict the processing of your personal data under certain conditions.
- **Right to Object to Processing:** You have the right to object to our processing of your personal data under certain conditions.
- **Right to Data Portability:** You have the right to request that we transfer the data that we have collected to another organization, or directly to you, under certain conditions.
- **Right to Withdraw Consent:** If we are relying on your consent to process your data, you have the right to withdraw that consent at any time.

To exercise any of these rights, please contact us using the details provided below.

7. Cookies and Similar Technologies

- **Local Storage (Anonymous User ID):** Our website uses localStorage to store your anonymous gospel_user_id. This is a browser-side storage mechanism, similar to a cookie, but it is not sent with every HTTP request. It helps us provide consistent functionality (like remembering your language preference) and contributes to anonymous usage statistics. **By continuing to use our website, you consent to the storage of this anonymous ID in your local storage.**
- **Google AdSense Cookies:** Google AdSense, as a third-party advertising service,

may use cookies and similar technologies (e.g., web beacons) to collect information about your visits to this and other websites to provide targeted advertisements. These cookies are controlled by Google. You can manage your Google Ad settings or opt out of personalized advertising by visiting Google's Ad Settings.

- **Cookie Consent Banner:** To comply with GDPR, we will implement a **cookie consent banner** on your first visit to the website. This banner will inform you about the use of local storage for the anonymous User ID and third-party cookies (like Google AdSense) and will require your explicit consent before these are activated. You will have the option to accept all, reject all (which may impact functionality like ad display), or manage your preferences.

8. Security Measures

We implement reasonable technical and organizational measures to protect your information from unauthorized access, disclosure, alteration, or destruction. This includes leveraging Cloudflare's robust security infrastructure, including DDoS protection and Turnstile bot verification.

9. Changes to This Privacy Policy

We may update our Privacy Policy from time to time. We will notify you of any changes by posting the new Privacy Policy on this page and updating the "Effective Date" at the top. You are advised to review this Privacy Policy periodically for any changes.

10. Contact Us

If you have any questions about this Privacy Policy or our data practices, please contact us at:

[Your Contact Email Address, e.g., privacy@[yourwebsitename].com]
[Your Company Name (if applicable)]
[Your Company Address (if applicable)]