Bethel Family Choir Website - Technical Documentation

Version 2.0 - June 2026

This technical documentation provides an in-depth overview of the architecture, components, and implementation details of the Bethel Family Choir website. It is intended for developers, maintainers, and contributors working on the project.

Table of Contents

1. System Overview

The Bethel Family Choir website is a comprehensive content management system (CMS) built for a Christian choir organization. It consists of a public-facing website for visitors and an admin dashboard for content management.

Key Features

- **Public Website**: Responsive design with dynamic content including hero sliders, gallery, history, services, and contact information
- **Admin Dashboard**: Secure authentication with full CRUD operations for all website content
- **Content Management**: Manage hero sections, scriptures, president messages, gallery images, songs, committees, and achievements
- **Media Management**: Upload and organize images and videos
- **Responsive Design**: Optimized for desktop, tablet, and mobile devices

Technology Stack

- **Frontend**: HTML5, CSS3, JavaScript (ES6+)
- **Backend**: PHP 8.0+, MySQL 5.7+
- **Libraries**: Font Awesome, Google Fonts
- **Architecture**: RESTful API with AJAX communication

2. Architecture

High-Level Architecture

```
┌───────────────┐    AJAX/Fetch     ┌───────────────┐
│   Frontend    │◄─────────────────►│  Backend API  │
│  (HTML/CSS/JS)│                   │  (PHP)        │
└───────────────┘                   └───────────────┘
        │                                   │
        │                                   │
        ▼                                   ▼
┌───────────────┐                   ┌───────────────┐
│   Browser     │                   │  Database     │
│   (Client)    │                   │  (MySQL)      │
└───────────────┘                   └───────────────┘
```

Data Flow

1. User interacts with frontend (clicks, form submissions)
2. JavaScript makes AJAX calls to PHP API endpoints
3. PHP processes requests, interacts with MySQL database
4. Data is returned as JSON to frontend
5. Frontend updates UI dynamically

3. Frontend Components

Core Files Structure

```
javascript/
├── main.js             # Mobile navigation and core utilities
├── home.js             # Homepage-specific functionality
├── about.js            # About page interactions
├── service.js          # Service page functionality
├── gallery.js          # Gallery page features
├── history.js          # History page components
└── api/
    └── renderApi.js    # API interaction and data rendering
```

Key Functions in renderApi.js

Slideshow()

```
async function Slideshow()
```

- **Purpose**: Loads and displays hero background images
- **API Call**: `fetchData("Slideshow")`
- **Parameters**: None
- **Returns**: Updates DOM with slider images
- **Error Handling**: Logs errors, continues with empty slideshow

HeroContent(targetpage)

```
async function HeroContent(targetpage)
```

- **Purpose**: Loads and renders page-specific hero content
- **API Call**: `fetchData("CTA", targetpage)`
- **Parameters**: *targetpage* (string) - Page identifier ("Home", "Service", etc.)
- **Returns**: Updates hero heading and caption with highlighting
- **Logic**: Alternates word highlighting for visual effect

BibleVerse()

```
async function BibleVerse()
```

- **Purpose**: Displays daily scripture
- **API Call**: `fetchData("Scripture")`
- **Parameters**: None
- **Returns**: Updates scripture display with icon and content

renderGallery()

```
async function renderGallery()
```

- **Purpose**: Shows gallery preview on homepage
- **API Call**: `fetchData("picture")`
- **Parameters**: None
- **Returns**: Displays first 4 gallery images as links

PresidentMessage()

```
async function PresidentMessage()
```

- **Purpose**: Shows president's message
- **API Call**: `fetchData("PresidentWord")`
- **Parameters**: None
- **Returns**: Renders profile image and message content

renderVideoGallery()

```
async function renderVideoGallery()
```

- **Purpose**: Displays YouTube video embeds
- **API Call**: `fetchData("song")`
- **Parameters**: None
- **Returns**: Creates responsive video cards with iframe embeds

CommitteeSlideshow()

```
async function CommitteeSlideshow()
```

- **Purpose**: Shows committee history slideshow
- **API Call**: `fetchData("Committee")`
- **Parameters**: None
- **Returns**: Creates interactive committee slides with images and member lists

createYearDetails()

```
async function createYearDetails()
```

- **Purpose**: Renders annual achievements timeline
- **API Call**: `fetchData("AnnualAchievement")`
- **Parameters**: None
- **Returns**: Creates chronological list of yearly achievements

API Client Functions (api.js)

fetchData(resource, id)

```
async function fetchData(resource, id = '')
```

- **Purpose**: Generic GET request handler
- **Parameters**:
  - *resource* (string): API endpoint name
  - *id* (string, optional): Resource identifier
- **Returns**: Promise resolving to data array or throwing error
- **Error Handling**: Shows user-friendly error messages

postData(resource, payload, id)

```
async function postData(resource, payload, id = '')
```

- **Purpose**: Generic POST request handler for creating resources
- **Parameters**:
  - *resource* (string): API endpoint name
  - *payload* (FormData): Form data to send
  - *id* (string, optional): Resource identifier for updates
- **Returns**: Promise with success/error response
- **Error Handling**: Displays success/error messages to user

4. Backend Components

Core Files Structure

```
admin/backend/
├── main.php            # Main API router and request handler
├── manager.php         # Database operations and business logic
└── api.js              # Frontend API client (already covered)
```

main.php - API Router

Architecture: Single entry point for all API requests using query parameter routing.

Request Routing Logic

```
$action = $_GET['action'] ?? '';
$id = $_GET['id'] ?? '';

switch ($action) {
    case 'login': // Authentication
    case 'CTA': // Call-to-action content
    case 'Scripture': // Daily scripture
    case 'PresidentWord': // President's message
    case 'song': // Music content
    case 'picture': // Gallery images
    case 'Slideshow': // Hero images
    case 'Committee': // Committee data
    case 'AnnualAchievement': // Yearly achievements
    case 'flyer': // Service flyers
    // ... more cases
}
```

HTTP Method Handling

- `GET` : Data retrieval
- `POST` : Create/update operations
- `DELETE` : Resource deletion
- `OPTIONS` : CORS preflight handling

manager.php - Database Operations

Class Structure:

```
class Manager {
    private $conn; // Database connection

    // Content management methods
    public function getCTA($page) { /* ... */ }
    public function updateCTA($data) { /* ... */ }
    public function getScripture($id) { /* ... */ }
    // ... more methods
}
```

Key Methods

Content Retrieval Methods

- `getCTA($page)` : Fetches hero content for specific page
- `getScripture($id)` : Retrieves daily scripture
- `getWord($id)` : Gets president's message
- `getPicture($id)` : Gallery image data
- `getSong($id)` : Music content
- `getCommittee($id)` : Committee information
- `getAnnualAchievements($id)` : Yearly achievements

Content Update Methods

- `updateCTA($data)` : Updates hero content
- `updateScripture($data)` : Modifies scripture
- `updateWord($data)` : Changes president's message
- `addPicture($data)` : Creates new gallery image
- `updatePicture($id, $data)` : Modifies existing image
- `addCommittee($data)` : Creates committee record
- `updateCommittee($id, $data)` : Updates committee info

5. API Reference

Authentication Endpoints

POST /admin/backend/main.php?action=login

Purpose: User authentication

Request Body:

```
{
  "email": "admin@example.com",
  "password": "password123"
}
```

Response:

```
{
  "success": true,
  "message": "Login successful",
  "data": { "user_id": 1, "email": "admin@example.com" }
}
```

GET /admin/backend/main.php?action=CTA&id={page}

Purpose: Get hero content for specific page

Parameters: `page` (Home, Service, Gallery, History, About)

Response:

```
{
  "success": true,
  "data": {
    "heading": "Welcome to Bethel",
    "caption": "Glorifying God through music"
  }
}
```

GET /admin/backend/main.php?action=Scripture

Purpose: Get daily scripture

Response:

```
{
  "success": true,
  "data": {
    "title": "John 3:16",
    "content": "For God so loved the world..."
  }
}
```

GET /admin/backend/main.php?action=picture

Purpose: Get gallery images

Response:

```
{
  "success": true,
  "data": [
    {
      "id": 1,
      "caption": "Choir performance",
      "link": "media/image1.jpg"
    }
  ]
}
```

GET /admin/backend/main.php?action=Committee

Purpose: Get committee information

Response:

```
{
  "success": true,
  "data": [
    {
      "id": 1,
      "era": "2024-2025",
      "ecree": "Bethel niferme",
      "picture": "media/committee.jpg",
      "members": [
        {
          "name": "John Doe",
          "post": "President"
        }
      ]
    }
  ]
}
```

6. Database Schema

Core Tables

users

```
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

pictures

```
CREATE TABLE pictures (
    id INT PRIMARY KEY AUTO_INCREMENT,
    caption VARCHAR(255),
    link VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

songs

```
CREATE TABLE songs (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255),
    link VARCHAR(500),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

committees

```
CREATE TABLE committees (
    id INT PRIMARY KEY AUTO_INCREMENT,
    era VARCHAR(50),
    ecree VARCHAR(255),
    picture VARCHAR(255),
    members JSON,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

annual_achievements

```
CREATE TABLE annual_achievements (
    id INT PRIMARY KEY AUTO_INCREMENT,
    year VARCHAR(20),
    summary TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

7. Security Considerations

Authentication

- Session-based authentication
- Password hashing (assumed in User class)
- Admin-only access to dashboard

Input Validation

- Server-side validation in PHP
- Prepared statements to prevent SQL injection
- File upload validation (type, size, malware)

CORS Configuration

- Restricted to specific origins in production
- Proper preflight handling

File Upload Security

- File type validation
- Unique filename generation
- Directory traversal protection

8. Deployment Guide

Prerequisites

- PHP 8.0+
- MySQL 5.7+
- Apache/Nginx web server
- Composer (for PHP dependencies)

Steps

1. **Clone Repository**

```
git clone [repository-url]
cd bethel-family-choir
```

2. **Configure Database**

```
CREATE DATABASE bethelfinal;
-- Import schema from schema.sql
```

3. **Update Configuration**
   - Edit `admin/backend/main.php` with database credentials
   - Configure CORS origins for production
4. **Set File Permissions**

```
chmod 755 media/
chown www-data:www-data media/
```

5. **Deploy Files**
   - Upload all files to web root
   - Ensure PHP files are executable
6. **Test Installation**
   - Access homepage
   - Test admin login
   - Verify API endpoints

9. Troubleshooting

Common Issues

API Returns 500 Error

- Check PHP error logs
- Verify database connection
- Ensure all required PHP extensions are installed

Images Not Uploading

- Check file permissions on media/ directory
- Verify upload_max_filesize in php.ini
- Check file type restrictions

JavaScript Errors

- Check browser console for errors
- Verify API_BASE_URL in api.js
- Ensure PHP server is running on correct port

Database Connection Failed

- Verify database credentials
- Check MySQL service status
- Ensure database exists

Debug Mode

Enable debug logging by setting:

```
ini_set('display_errors', 1);
error_reporting(E_ALL);
```

Performance Optimization

- Enable PHP opcode caching (OPcache)
- Use database indexes on frequently queried columns
- Implement CDN for static assets
- Compress images before upload

Note: This documentation provides a comprehensive technical overview for developers working on the Bethel Family Choir website. For specific implementation details, refer to the inline code comments and function documentation within the source files.

if you have any questions or need further assistance, please contact the development team at sezeranoshimo@gmail.com

or visit our website at www.tristechhub.org.rw