CLAUDE

CLAUDE.md

This file provides guidance to Claude Code (claude.ai/code) when working with code in this repository.

Related Documentation

- Content Rotation System - Detailed guide for configuring and troubleshooting the content rotation system

Current Work: Enhanced Featured Content System (2025-10-30)

Overview

Implementing an improved featured content system with: 1. Time-based decay for meeting relevance 2. AI engagement score integration 3. Daytime scheduling priority 4. Separate replay delays for featured vs regular content

Implementation Plan

Phase 1: Configuration Schema Updates [DONE] (In Progress)

- Add `featured_content` section to scheduling config
- Add `meeting_relevance` configuration
- Add `content_priorities` by type
- Ensure backward compatibility

Phase 2: Meeting Decay Logic (Pending)

- Modify `get_featured_content()` in scheduler_postgres.py
- Calculate meeting age from meeting_date
- Apply relevance tiers (fresh/relevant/archive)
- Adjust featured status based on age

Phase 3: Daytime Priority (Pending)

- Add time-of-day check to scheduling logic
- Implement 75% daytime probability for featured content
- Use existing timeslot configuration

Phase 4: AI Engagement Integration (Pending)

- Check for engagement_score in scheduling_metadata
- Auto-feature high engagement content
- Apply thresholds by content type

Phase 5: Frontend Updates (Pending)

- Add UI controls for featured content settings
- Display meeting age and relevance status
- Show featured content metrics in reports

Configuration Structure

```json
{
  "scheduling": {
    "featured_content": {
      "daytime_hours": {"start": 6, "end": 18},
      "daytime_probability": 0.75,
      "minimum_spacing": 2,
      "meeting_decay": true
    },
    "meeting_relevance": {
      "fresh_days": 3,
      "relevant_days": 7,
      "archive_days": 14,
      "expire_after": 18
    },
    "content_priorities": {
      "MTG": {
        "auto_feature_days": 3,
        "engagement_threshold": 70,
        "max_daily_plays": {
          "fresh": 4,
          "relevant": 2,
          "archive": 1
        }
      }
    },
    "replay_delays": {
      "featured": {
        "id": 2,
        "spots": 3,
        "short_form": 4,
        "long_form": 6
      },
      "regular": {
        "id": 24,
        "spots": 48,
        "short_form": 72,
        "long_form": 72
      }
    }
  }
}
```

Progress Tracking

- ☐ Update config_manager.py defaults
- ☐ Modify scheduler_postgres.py featured content logic
- ☐ Add meeting age calculation
- ☐ Implement daytime scheduling bias
- ☐ Test with weekly schedules
- ☐ Update documentation

Development Commands

Running the Application

```
# Start the backend server (Flask API)
cd backend && python app.py

# Start the frontend server (static file server)
cd frontend && python frontend_server.py
```

Python Environment

```
# Install dependencies
pip install -r backend/requirements.txt

# Dependencies include:
# - Flask==2.3.3
# - Flask-CORS==4.0.0
# - requests==2.31.0
# - python-dateutil==2.8.2
# - python-dotenv==1.0.0
```

Environment Variables

API keys are stored in environment variables for security:

```
# Copy the example environment file
cp .env.example .env

# Edit .env with your actual API keys
# OPENAI_API_KEY=your_openai_api_key_here
# ANTHROPIC_API_KEY=your_anthropic_api_key_here
```

Important: Never commit the `.env` file to version control. It's already included in `.gitignore`.

Testing

There are no automated tests in this codebase. Testing is done manually through the web interface.

Architecture Overview

This is a Flask-based FTP media synchronization application with a Python backend and HTML/CSS/JavaScript frontend.

Core Components

Backend (Flask API - Port 5000) - `app.py` - Main Flask application with API endpoints - `config_manager.py` - Configuration management and JSON persistence - `file_scanner.py` - FTP directory scanning and file filtering - `ftp_manager.py` - FTP connection management using Python's ftplib

Frontend (Static Server - Port 8000) - `frontend_server.py` - Simple Flask static file server - `index.html` - Main UI with server configuration and file management - `script.js` - Frontend logic for API calls and UI interactions - `styles.css` - Application styling

Key API Endpoints

- `GET /api/config` - Retrieve current configuration
- `POST /api/config` - Save configuration
- `POST /api/test-connection` - Test FTP connection
- `POST /api/scan-files` - Scan FTP directories for files
- `POST /api/sync-files` - Synchronize files between servers

FTP Manager Implementation

The `FTPManager` class provides FTP connection management with methods: - `test_connection()` - Test FTP connection and disconnect - `connect()` - Establish persistent FTP connection - `disconnect()` - Close FTP connection - `list_files(path)` - List files in directory with size and permissions - `download_file(remote_path, local_path)` - Download file from FTP server - `upload_file(local_path, remote_path)` - Upload file to FTP server with directory creation - `copy_file_to(file_info, target_ftp, keep_temp=False)` - Copy files between FTP servers - `update_file_to(file_info, target_ftp, keep_temp=False)` - Update files (same as copy) - `create_directory(path)` - Create directory structure on FTP server - Properties: `connected`, `ftp` (for direct ftplib access)

Configuration System

- Configuration stored in `config.json` (auto-generated)
- Sample configuration available in `config.sample.json`
- Supports source/target server configs, sync settings, and UI preferences
- Passwords are not saved to config files for security

File Filtering

The application filters files based on: - File extensions (mp4, mkv, avi, mov, wmv, etc.) - File size limits (min/max) - Subdirectory inclusion - Overwrite policies

Scheduling System

The application includes a sophisticated content scheduling system: - **PostgreSQL-based scheduler** (`scheduler_postgres.py`) - Manages schedule creation and content rotation - **Content rotation system** - Cycles through duration categories (ID, SPOTS, SHORT_FORM, LONG_FORM) - **Template-based scheduling** - Import/export daily, weekly, and monthly templates - **Gap filling** - Automatically fills empty time slots with appropriate content - **Replay delays** - Prevents content from repeating too frequently - **Content expiration** - Manages content lifecycle with expiration dates

Key scheduling features: - Create daily/weekly schedules - Import meeting schedules from PDFs - Fill template gaps with rotation-aware content selection - Export schedules in various formats - Track content usage and prevent over-rotation

See CONTENT_ROTATION.md for detailed rotation configuration.

Important Notes

- The application requires two FTP servers configured as “source” and “target”
- All file operations support dry-run mode for testing
- Frontend runs on 127.0.0.1:8000, backend on 127.0.0.1:5000
- CORS is configured to allow frontend-to-backend communication
- Detailed logging is enabled for debugging FTP operations
- The application is designed for media file synchronization with size-based filtering