Interface Wizard Backend API Documentation

Version: 2.0.0

Date: December 20, 2025

Author: Interface Wizard Development Team

File: `mainwithfastapi.py`

Table of Contents

Executive Summary

What is Interface Wizard?

Interface Wizard is a healthcare integration platform that:

- **Converts natural language** into HL7 v2.x messages
- **Processes CSV/Excel files** containing patient data
- **Generates valid HL7 messages** automatically
- **Sends messages to Mirth Connect** via MLLP protocol
- **Provides real-time progress updates** during processing

Key Features

■ **AI-Powered Generation** - Uses OpenAI GPT-4 or fallback generator

■ **Real-Time Progress** - Server-Sent Events (SSE) for live updates

■ **Dashboard Statistics** - Track processed patients and success rates

■ **Flexible CSV Parsing** - Handles various column naming conventions

■ **Mirth Integration** - MLLP protocol for HL7 message transmission

■ **Validation System** - 3-tier field validation (System, Contextual, Critical)

■ **Dual Mode Operation** - API mode (FastAPI) + Console mode

Who Should Use This Document?

- **Backend Developers** - Understanding the API implementation
- **Frontend Developers** - Integrating with the API endpoints
- **System Administrators** - Deploying and maintaining the system
- **Healthcare IT** - Understanding HL7 message generation
- **QA Engineers** - Testing the complete workflow
- **Project Managers** - Understanding system capabilities

System Architecture

High-Level Overview

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■                         USER INTERFACE                           ■
■                    (Frontend - Angular/React)                    ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                             ■
                             ■ HTTP/REST API
                             ■ Server-Sent Events (SSE)
                             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■                      FASTAPI BACKEND                             ■
■                   (main_with_fastapi.py)                         ■
■                                                                   ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■    ■
■  ■  API Endpoints Layer                                     ■    ■
■  ■  - Dashboard endpoints                                   ■    ■
■  ■  - Real-time upload endpoints                           ■    ■
■  ■  - Legacy bulk processing endpoints                     ■    ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■    ■
■                             ■                                     ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■    ■
■  ■  Business Logic Layer                                    ■    ■
■  ■  - CSV parsing & column mapping                         ■    ■
■  ■  - Patient data extraction                              ■    ■
■  ■  - HL7 message generation                               ■    ■
■  ■  - Validation (3-tier system)                           ■    ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■    ■
■                             ■                                     ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■    ■
■  ■  Integration Layer                                       ■    ■
■  ■  - OpenAI API (GPT-4o-mini)                             ■    ■
■  ■  - MLLP Protocol Handler                                ■    ■
■  ■  - Session Management (In-Memory)                       ■    ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■    ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                             ■
                             ■ MLLP over TCP/IP
                             ■ (Port 6661)
                             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■                      MIRTH CONNECT                               ■
■                    (HL7 Message Router)                          ■
■                                                                   ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■    ■
■  ■  Channel: IW Listener HL7                               ■    ■
■  ■  - Receives HL7 messages via MLLP                       ■    ■
■  ■  - Validates message structure                          ■    ■
■  ■  - Transforms & routes messages                         ■    ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■    ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                             ■
                             ■ Database Insert
                             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■                      OPENEMR DATABASE                            ■
■                    (MySQL - patient_data)                        ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

Component Responsibilities

Component	Responsibility	Technology
Frontend	User interface, file upload, real-time display	Angular 17 / React 18

Component	Responsibility	Technology
FastAPI Backend	REST API, HL7 generation, validation	Python 3.9+, FastAPI
OpenAI API	AI-powered HL7 message generation	GPT-4o-mini
Mirth Connect	HL7 message routing and transformation	Mirth Connect 4.x
OpenEMR	Patient data storage	MySQL 8.0+

Technology Stack

Backend Technologies

```
Language: Python 3.9+
Web Framework: FastAPI 0.109.0
ASGI Server: Uvicorn 0.27.0
Data Validation: Pydantic 2.5.3
HL7 Parsing: python-hl7 0.4.5
Data Processing: Pandas 2.1.4
Excel Support: openpyxl 3.1.2
AI Integration: OpenAI 1.10.0
File Upload: python-multipart 0.0.6
```

Key Python Libraries Explained

****FastAPI****

- Modern, high-performance web framework
- Automatic API documentation (Swagger UI)
- Built-in validation with Pydantic
- Async/await support for real-time features

****Uvicorn****

- Lightning-fast ASGI server
- Handles concurrent connections
- WebSocket and SSE support

****Pydantic****

- Data validation using Python type hints
- Automatic JSON serialization/deserialization
- Clear error messages for invalid data

****python-hl7****

- HL7 v2.x message parsing and validation
- Segment and field extraction
- Message structure validation

****Pandas****

- CSV/Excel file reading
- Data transformation and cleaning
- Flexible column mapping

****OpenAI****

- GPT-4o-mini integration
- Natural language to HL7 conversion
- Fallback to deterministic generator if unavailable

Complete Architecture Diagrams

1. Real-Time CSV Upload Workflow

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  STEP 1: User Uploads CSV File                                   ■
■  Frontend → POST /api/upload                                      ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                              ■
                              ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  STEP 2: Backend Creates Session                                 ■
■  - Generate unique upload_id (UUID)                              ■
■  - Store in upload_sessions dictionary                           ■
■  - Return upload_id to frontend immediately                      ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                              ■
                              ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  STEP 3: Frontend Connects to SSE Stream                         ■
■  GET /api/upload/{upload_id}/stream                              ■
■  EventSource connection established                              ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

```
                             ■
                             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  STEP 4: Background Task Starts (asyncio)                        ■
■  async def process_csv_with_progress()                           ■
■                                                                    ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■  ■ Step 1: File Uploaded ✓                                 ■   ■
■  ■ - Mark current_step = 1                                 ■   ■
■  ■ - Update step_status = "File uploaded successfully"   ■   ■
■  ■ - Wait 0.5s                                             ■   ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■                                                                    ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■  ■ Step 2: Parse CSV Data                                  ■   ■
■  ■ - Extract patient data from CSV                       ■   ■
■  ■ - Map flexible column names                           ■   ■
■  ■ - Normalize dates, addresses                          ■   ■
■  ■ - Store parsed_patients array                         ■   ■
■  ■ - Update total_patients count                         ■   ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■                                                                    ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■  ■ Step 3: Select Patients (Auto-select all)             ■   ■
■  ■ - Mark all patients as selected                       ■   ■
■  ■ - Store selected_patient_ids array                    ■   ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■                                                                    ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■  ■ Step 4: Generate HL7 Messages                           ■   ■
■  ■ - Loop through each patient                           ■   ■
■  ■ - Call OpenAI API or fallback generator             ■   ■
■  ■ - Validate generated HL7 message                      ■   ■
■  ■ - Update progress: "Generated 3/10 messages"          ■   ■
■  ■ - Rate limiting: 1 second delay between messages      ■   ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■                                                                    ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■  ■ Step 5: Send to Mirth (if enabled)                      ■   ■
■  ■ - Loop through generated messages                       ■   ■
■  ■ - Send via MLLP protocol (TCP socket)                 ■   ■
■  ■ - Wait for ACK/NACK response                            ■   ■
■  ■ - Track successful/failed sends                       ■   ■
■  ■ - Update progress: "Sent 5/10 to Mirth"               ■   ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■                                                                    ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■  ■ Step 6: Complete                                          ■   ■
■  ■ - Mark status = "completed"                             ■   ■
■  ■ - Set completed_at timestamp                            ■   ■
■  ■ - Update dashboard statistics                           ■   ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                             ■
                             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  STEP 5: SSE Stream Sends Updates Every 500ms                    ■
■                                                                    ■
■  data: {"current_step":1,"step_status":"File uploaded",...}    ■
■  data: {"current_step":2,"step_status":"Parsing CSV...",...}   ■
■  data: {"current_step":4,"step_status":"Generated 3/10",...}   ■
■  data: {"current_step":6,"status":"completed",...}             ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                             ■
                             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  STEP 6: Frontend Updates UI in Real-Time                        ■
■  - Highlight current step (1-6)                                  ■
■  - Show status message                                           ■
■  - Update progress bar (3/10, 5/10, etc.)                        ■
■  - Display completion screen with results                        ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

2. HL7 Message Generation Flow

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  INPUT: Patient Data from CSV                                  ■
■  {                                                               ■
■    patient_id: "PAT001",                                       ■
■    first_name: "John",                                         ■
■    last_name: "Doe",                                           ■
■    dob: "1990-01-15",                                          ■
■    gender: "M",                                                  ■
■    address: "123 Main St, Boston, MA 02101"                   ■
■  }                                                               ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                             ■
                             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  STEP 1: Build Command Text                                    ■
■  "Trigger Event: ADT-A01                                       ■
■   Patient ID: PAT001                                           ■
■   Patient Name: Doe John                                       ■
■   Date of Birth: 19900115                                      ■
■   Gender: M                                                      ■
■   Address: 123 Main St, Boston, MA 02101                      ■
■   Create an ADT-A01 message for patient..."                   ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                             ■
                             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  STEP 2: Call HL7 Generator                                    ■
■  generate_hl7_message(client_wrapper, command)                 ■
■                                                                  ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■     ■■■■■■■■■■■■■■■■■■■■■■■■■■■■     ■
■  ■  Try: OpenAI API      ■■■■■■■  Success: Use AI result ■     ■
■  ■  (GPT-4o-mini)        ■     ■■■■■■■■■■■■■■■■■■■■■■■■■■■■     ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■                                     ■
■            ■                                                     ■
■            ■ Fail (no API key, rate limit, error)              ■
■            ▼                                                     ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■     ■■■■■■■■■■■■■■■■■■■■■■■■■■■■     ■
■  ■  Fallback Generator   ■■■■■■■  Use deterministic      ■     ■
■  ■  (Local/Offline)      ■     ■  HL7 generation        ■     ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■     ■■■■■■■■■■■■■■■■■■■■■■■■■■■■     ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                             ■
                             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  STEP 3: Generated HL7 Message (HL7 v2.5)                      ■
■                                                                  ■
■  MSH|^~\&|SMART_APP|SMART_FAC|REC_APP|REC_FAC|20251220140030||  ■
■      ADT^A01|abc-123-def-456|P|2.5                             ■
■  EVN|A01|20251220140030                                        ■
■  PID|1||PAT001||Doe^John||19900115|M|||123 Main St, Boston,   ■
■      MA 02101                                                    ■
■  PV1|1|I                                                         ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                             ■
                             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  STEP 4: Validate HL7 Message (3-Tier System)                  ■
■                                                                  ■
■  Tier 1: System Fields (Auto-populated - always valid)        ■
■  ✓ MSH-11 (Processing ID) = "P"                                ■
■  ✓ MSH-12 (Version) = "2.5"                                    ■
■  ✓ EVN-2 (Timestamp) = "20251220140030"                        ■
■                                                                  ■
■  Tier 2: Contextual Fields (Smart defaults - always valid)    ■
■  ✓ PV1-2 (Patient Class) = "I" (inferred from ADT-A01)       ■
■                                                                  ■
■  Tier 3: Critical Fields (User data - MUST be present)        ■
■  ✓ MSH-9 (Message Type) = "ADT^A01"                             ■
■  ✓ PID-3 (Patient ID) = "PAT001"                                ■
■  ✓ PID-5 (Patient Name) = "Doe^John"                            ■
■  ✓ PID-7 (DOB) = "19900115"                                     ■
■  ✓ PID-8 (Gender) = "M"                                          ■
■                                                                    ■
■  Result: ■ VALIDATION PASSED                                    ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                             ■
                             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  STEP 5: Send to Mirth Connect (Optional)                        ■
■                                                                    ■
■  MLLP Envelope:                                                   ■
■  <VT>HL7_MESSAGE<FS><CR>                                         ■
■                                                                    ■
■  Where:                                                           ■
■  <VT>  = 0x0B (Start Block)                                      ■
■  <FS>  = 0x1C (End Block)                                        ■
■  <CR>  = 0x0D (Carriage Return)                                  ■
■                                                                    ■
■  TCP Socket → localhost:6661                                      ■
■  Wait for ACK/NACK response                                      ■
■                                                                    ■
■  Response: MSA|AA|abc-123-def-456                                ■
■  Result: ■ Message accepted by Mirth                            ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

3. Server-Sent Events (SSE) Architecture

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  FRONTEND: EventSource Connection                                ■
■  const eventSource = new EventSource('/api/upload/123/stream'); ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                             ■
                             ■ HTTP GET (Keep-Alive Connection)
                             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  BACKEND: SSE Endpoint                                            ■
■  @app.get("/api/upload/{upload_id}/stream")                      ■
■                                                                    ■
■  async def event_generator():                                    ■
■      while session["status"] == "processing":                    ■
■          # Build event data from session                         ■
■          event_data = {                                           ■
■              "current_step": session["current_step"],            ■
■              "step_status": session["step_status"],              ■
■              "total_patients": session["total_patients"]         ■
■          }                                                         ■
■                                                                    ■
■          # Send SSE event                                         ■
■          yield f"data: {json.dumps(event_data)}\n\n"            ■
■                                                                    ■
■          # Wait 500ms before next update                         ■
■          await asyncio.sleep(0.5)                                ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                             ■
                             ■ SSE Stream (text/event-stream)
                             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  NETWORK: SSE Events                                             ■
■                                                                    ■
■  data: {"current_step":1,"step_status":"File uploaded",...}    ■
■  (500ms pause)                                                    ■
■  data: {"current_step":2,"step_status":"Parsing CSV...",...}   ■
■  (500ms pause)                                                    ■
■  data: {"current_step":3,"step_status":"Selected 10",...}      ■
■  (500ms pause)                                                   ■
■  data: {"current_step":4,"step_status":"Generated 1/10",...}   ■
■  (500ms pause)                                                   ■
■  data: {"current_step":4,"step_status":"Generated 2/10",...}   ■
■  ...                                                             ■
■  data: {"current_step":6,"status":"completed",...}             ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                             ■
                             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  FRONTEND: EventSource.onmessage Handler                         ■
■                                                                    ■
■  eventSource.onmessage = (event) => {                            ■
■      const data = JSON.parse(event.data);                        ■
■                                                                    ■
■      // Update wizard UI                                          ■
■      setActiveStep(data.current_step);  // Highlight 1-6        ■
■      setStatusMessage(data.step_status); // Show status         ■
■      updateProgressBar(data.generated_count);                    ■
■                                                                    ■
■      // Close connection when done                               ■
■      if (data.status === 'completed') {                          ■
■          eventSource.close();                                    ■
■          showResults(upload_id);                                 ■
■      }                                                             ■
■  };                                                               ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

4. In-Memory Data Storage Structure

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  PYTHON DICTIONARIES (In-Memory Storage)                         ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  upload_sessions: Dict[str, Dict[str, Any]]                      ■
■                                                                    ■
■  {                                                                 ■
■    "abc-123-def-456": {                                          ■
■      "id": "abc-123-def-456",                                    ■
■      "filename": "patients.csv",                                 ■
■      "status": "processing",  // or "completed", "error"       ■
■      "current_step": 4,        // 1-6                            ■
■      "step_status": "Generated 5/10 messages",                   ■
■      "total_patients": 10,                                       ■
■      "created_at": "2025-12-20T14:00:00",                        ■
■      "completed_at": null,     // or ISO timestamp               ■
■      "trigger_event": "ADT-A01",                                 ■
■      "send_to_mirth": true,                                      ■
■                                                                    ■
■      // Step 2: Parsed patient data                              ■
■      "parsed_patients": [                                         ■
■        {                                                           ■
■          "row_number": 2,                                         ■
■          "patient_id": "PAT001",                                 ■
■          "first_name": "John",                                   ■
■          "last_name": "Doe",                                     ■
■          "patient_name": "John Doe",                             ■
■          "dob": "19900115",                                      ■
■          "gender": "M",                                           ■
■          "address": "123 Main St, Boston, MA 02101",           ■
■          "selected": true                                         ■
■        },                                                          ■
■        ...                                                         ■
■      ],                                                            ■
■                                                                    ■
■      // Step 3: Selected patient IDs                             ■
■      "selected_patient_ids": ["PAT001", "PAT002", ...],        ■
■                                                                    ■
■      // Step 4: Generated HL7 messages                           ■
■      "generated_messages": [                                      ■
■        {                                                           ■
■          "row_number": 2,                                         ■
■          "patient_id": "PAT001",                                 ■
■          "patient_name": "John Doe",                             ■
■          "hl7_message": "MSH|^~\\&|SMART_APP|...",               ■
■          "validation": {                                          ■
■            "is_valid": true,                                     ■
■            "missing_fields": [],                                 ■
■            "message": "Validation passed"                        ■
■          },                                                        ■
■          "status": "success",                                     ■
■          "mirth_sent": true,      // if send_to_mirth enabled   ■
■          "mirth_ack": "MSA|AA|..."  // ACK from Mirth            ■
■        },                                                          ■
■        ...                                                         ■
■      ],                                                            ■
■                                                                    ■
■      // Step 5: Mirth send statistics                            ■
■      "mirth_successful": 9,                                      ■
■      "mirth_failed": 1                                            ■
■    },                                                              ■
■                                                                    ■
■    "xyz-789-ghi-012": { ... },  // Another session               ■
■    ...                                                             ■
■  }                                                                 ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  dashboard_stats: Dict[str, int]                                 ■
■                                                                    ■
■  {                                                                 ■
■    "total_processed": 150,         // Total patients             ■
■    "hl7_messages_generated": 148,  // Total messages created     ■
■    "successful_sends": 145,        // Successfully sent          ■
■    "failed_sends": 3               // Failed to send             ■
■  }                                                                 ■
■                                                                    ■
■  // Updated automatically as processing completes                ■
■  // Used by GET /api/dashboard/stats                             ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

Note: Data persists only while server is running
      Restart = all sessions lost (acceptable for dev)
      Production: Consider adding database persistence
```

API Endpoints Reference

Complete Endpoint List

Method	Endpoint	Tag	Purpose
GET	`/`	General	API information
GET	`/health`	General	Health check

Method	Endpoint	Tag	Purpose
GET	`/api/dashboard/stats`	Dashboard	Dashboard statistics
GET	`/api/dashboard/system-status`	Dashboard	System health
POST	`/api/upload`	Real-Time Upload	Upload CSV with real-time progress
GET	`/api/upload/{id}/stream`	Real-Time Upload	SSE progress stream
GET	`/api/upload/{id}/status`	Real-Time Upload	Current status (polling)
GET	`/api/upload/{id}/results`	Real-Time Upload	Final results
POST	`/api/generate-hl7`	HL7 Generation	Generate from text
POST	`/api/validate-hl7`	HL7 Validation	Validate HL7 message
POST	`/api/send-to-mirth`	Mirth Integration	Send single message
POST	`/api/upload-excel`	Bulk Processing	Legacy bulk upload
GET	`/api/supported-events`	Reference	Supported trigger events

---

1. GET /api/dashboard/stats

Purpose: Get dashboard statistics for UI tiles

Request:

```
GET /api/dashboard/stats HTTP/1.1
Host: localhost:8000
```

Response:

```
{
  "total_processed": 150,
  "hl7_messages": 148,
  "successful_sends": 145,
  "failed_sends": 3,
  "success_rate": 98.0
}
```

Response Fields:

- `total_processed` (integer) - Total patients processed across all sessions
- `hl7_messages` (integer) - Total HL7 messages generated
- `successful_sends` (integer) - Successfully sent to Mirth

- `failed_sends` (integer) - Failed to send to Mirth
- `success_rate` (float) - Success percentage (0-100)

Use Case:

```
// Frontend: Poll every 5 seconds to update dashboard
setInterval(async () => {
  const stats = await fetch('/api/dashboard/stats').then(r => r.json());

  document.getElementById('total-tile').textContent = stats.total_processed;
  document.getElementById('messages-tile').textContent = stats.hl7_messages;
  document.getElementById('success-rate-tile').textContent = stats.success_rate + '%';
}, 5000);
```

---

2. GET /api/dashboard/system-status

Purpose: Check system health for dashboard indicators

Request:

```
GET /api/dashboard/system-status HTTP/1.1
Host: localhost:8000
```

Response:

```
{
  "openemr_connection": {
    "status": "Active",
    "last_sync": "2025-12-20T14:30:00.123456"
  },
  "hl7_parser": {
    "status": "Running"
  },
  "message_queue": {
    "status": "Ready",
    "pending": 0
  }
}
```

Response Fields:

- `openemr_connection.status` - "Active" if Mirth reachable, "Offline" otherwise
- `openemr`*`connection.lastsync`* - Current timestamp (ISO format)
- `hl7_parser.status` - "Running" if OpenAI available, "Limited" if using fallback
- `message_queue.status` - Always "Ready" (for future queue implementation)

- `message_queue.pending` - Always 0 (for future queue implementation)

How It Works:

```
# Tests Mirth connection by attempting socket connection
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.settimeout(2)
result = sock.connect_ex((MIRTH_HOST, MIRTH_PORT))
sock.close()

if result == 0:
    mirth_status = "Active"  # Connection successful
else:
    mirth_status = "Offline"  # Connection failed
```

Use Case:

```
// Frontend: Show green/red indicators
const status = await fetch('/api/dashboard/system-status').then(r => r.json());

if (status.openemr_connection.status === 'Active') {
  showGreenDot('mirth-indicator');
} else {
  showRedDot('mirth-indicator');
}
```

---

3. POST /api/upload

Purpose: Upload CSV file and start background processing with real-time updates

Request:

```
POST /api/upload HTTP/1.1
Host: localhost:8000
Content-Type: multipart/form-data; boundary=----Boundary

------Boundary
Content-Disposition: form-data; name="file"; filename="patients.csv"
Content-Type: text/csv

Patient ID,First Name,Last Name,DOB,Gender,Address
PAT001,John,Doe,1990-01-15,M,123 Main St
------Boundary
Content-Disposition: form-data; name="trigger_event"

ADT-A01
------Boundary
Content-Disposition: form-data; name="send_to_mirth"

true
------Boundary--
```

Request Parameters:

- `file` (file, required) - CSV or Excel file with patient data
- `trigger_event` (string, optional) - HL7 trigger event (default: "ADT-A01")
- send*to*mirth (boolean, optional) - Send to Mirth after generation (default: false)

Response:

```
{
  "upload_id": "abc-123-def-456",
  "filename": "patients.csv",
  "status": "processing",
  "message": "Upload started. Connect to /api/upload/abc-123-def-456/stream for real-time
updates"
}
```

Response Fields:

- `upload_id` (string) - Unique session ID (UUID) - **Save this for next steps!**
- `filename` (string) - Original filename
- `status` (string) - Always "processing" on successful upload
- `message` (string) - Instructions for connecting to SSE stream

What Happens Next:

1. Session created with unique upload_id
2. CSV file read and parsed
3. Background async task starts (process*csv*with_progress)
4. Processing happens in background through 6 steps
5. Frontend should connect to `/api/upload/{upload_id}/stream` for updates

Use Case:

```
// Frontend: Upload file and get upload_id
async function uploadCSV(file) {
  const formData = new FormData();
  formData.append('file', file);
  formData.append('trigger_event', 'ADT-A01');
  formData.append('send_to_mirth', true);

  const response = await fetch('/api/upload', {
    method: 'POST',
    body: formData
  });

  const data = await response.json();
  console.log('Upload ID:', data.upload_id);

  // Now connect to SSE stream
  connectToProgressStream(data.upload_id);
}
```

4. GET /api/upload/{upload_id}/stream

Purpose: Stream real-time progress updates using Server-Sent Events (SSE)

Request:

```
GET /api/upload/abc-123-def-456/stream HTTP/1.1
Host: localhost:8000
Accept: text/event-stream
```

Response: (Server-Sent Events stream)

```
data: {"current_step":1,"step_status":"File uploaded successfully","total_patients":0,
"status":"processing"}

data: {"current_step":2,"step_status":"Parsing CSV data...","total_patients":10,"status":
"processing"}

data: {"current_step":3,"step_status":"Selected 10 patients","total_patients":10,"status":
"processing"}

data: {"current_step":4,"step_status":"Generated 1/10 messages","total_patients":10,
"status":"processing","generated_count":1}

data: {"current_step":4,"step_status":"Generated 2/10 messages","total_patients":10,
"status":"processing","generated_count":2}

data: {"current_step":4,"step_status":"Generated 5/10 messages","total_patients":10,
"status":"processing","generated_count":5}

data: {"current_step":5,"step_status":"Sending to Mirth Connect...","total_patients":10,
"status":"processing","mirth_successful":3,"mirth_failed":0}

data: {"current_step":6,"step_status":"Processing complete!","total_patients":10,"status":
"completed","upload_id":"abc-123-def-456","successful":10,"failed":0}
```

Event Data Fields:

- `current_step` (integer) - Current step number (1-6)
- `step_status` (string) - Human-readable status message
- `total_patients` (integer) - Total patient count
- `status` (string) - "processing", "completed", or "error"
- `generated_count` (integer, optional) - Messages generated (step 4)
- `mirth_successful` (integer, optional) - Successful Mirth sends (step 5)
- `mirth_failed` (integer, optional) - Failed Mirth sends (step 5)

- `upload_id` (string, final event) - Session ID
- `successful` (integer, final event) - Successful message count
- `failed` (integer, final event) - Failed message count
- `error` (string, if error) - Error message

Update Frequency: Every 500ms (0.5 seconds)

Use Case:

```
// Frontend: Connect to SSE stream and update UI
function connectToProgressStream(uploadId) {
  const eventSource = new EventSource(`/api/upload/${uploadId}/stream`);

  eventSource.onmessage = (event) => {
    const data = JSON.parse(event.data);

    // Update wizard step indicator
    highlightStep(data.current_step);  // Highlight step 1-6

    // Update status message
    document.getElementById('status').textContent = data.step_status;

    // Update progress bar (if in step 4)
    if (data.generated_count && data.total_patients) {
      const percent = (data.generated_count / data.total_patients) * 100;
      document.getElementById('progress-bar').style.width = percent + '%';
      document.getElementById('progress-text').textContent =
        `${data.generated_count}/${data.total_patients}`;
    }

    // When complete, close connection and show results
    if (data.status === 'completed') {
      eventSource.close();
      loadResults(uploadId);
    }

    // Handle errors
    if (data.status === 'error') {
      eventSource.close();
      showError(data.error);
    }
  };

  eventSource.onerror = () => {
    console.error('SSE connection error');
    eventSource.close();
  };
}
```

React Example:

```
import { useEffect, useState } from 'react';

function UploadProgress({ uploadId }) {
  const [progress, setProgress] = useState({});

  useEffect(() => {
    const eventSource = new EventSource(`/api/upload/${uploadId}/stream`);

    eventSource.onmessage = (event) => {
      const data = JSON.parse(event.data);
      setProgress(data);
      if (data.status === 'completed' || data.status === 'error') {
        eventSource.close();
      }
    };

    return () => eventSource.close();
  }, [uploadId]);

  return (
    <div>
      <h3>Step {progress.current_step}/6</h3>
      <p>{progress.step_status}</p>
      {progress.generated_count && (
        <progress
          value={progress.generated_count}
          max={progress.total_patients}
        />
      )}
    </div>
  );
}
```

---

5. GET /api/upload/{upload_id}/status

Purpose: Get current upload status (polling alternative to SSE)

Request:

```
GET /api/upload/abc-123-def-456/status HTTP/1.1
Host: localhost:8000
```

Response:

```
{
  "upload_id": "abc-123-def-456",
  "filename": "patients.csv",
  "status": "processing",
  "current_step": 4,
  "step_status": "Generated 5/10 messages",
  "total_patients": 10,
  "created_at": "2025-12-20T14:00:00.123456",
  "completed_at": null
}
```

Response Fields:

- `upload_id` (string) - Session ID
- `filename` (string) - Original filename
- `status` (string) - "processing", "completed", or "error"

- `current_step` (integer) - Current step (1-6)
- `step_status` (string) - Status message
- `total_patients` (integer) - Total patient count
- `created_at` (string) - Upload timestamp (ISO format)
- `completed_at` (string | null) - Completion timestamp or null if still processing

Use Case (Polling Alternative):

```
// For environments that don't support SSE
async function pollStatus(uploadId) {
  const interval = setInterval(async () => {
    const status = await fetch(`/api/upload/${uploadId}/status`)
      .then(r => r.json());

    // Update UI
    updateStepIndicator(status.current_step);
    updateStatusMessage(status.step_status);

    // Stop polling when complete
    if (status.status === 'completed' || status.status === 'error') {
      clearInterval(interval);
      loadResults(uploadId);
    }
  }, 1000);  // Poll every 1 second
}
```

6. GET /api/upload/{upload_id}/results

Purpose: Get complete results after processing is finished

Request:

```
GET /api/upload/abc-123-def-456/results HTTP/1.1
Host: localhost:8000
```

Response:

```
{
  "upload_id": "abc-123-def-456",
  "filename": "patients.csv",
  "status": "completed",
  "current_step": 6,
  "total_patients": 10,
  "successful": 9,
  "failed": 1,
  "mirth_successful": 9,
  "mirth_failed": 0,
  "created_at": "2025-12-20T14:00:00.123456",
  "completed_at": "2025-12-20T14:00:23.789012",
  "messages": [
    {
      "row_number": 2,
      "patient_id": "PAT001",
      "patient_name": "John Doe",
      "hl7_message": "MSH|^~\\
&|SMART_APP|SMART_FAC|REC_APP|REC_FAC|20251220140005||ADT^A01|abc-123|P|2.5\
nEVN|A01|20251220140005\nPID|1||PAT001||Doe^John||19900115|M|||123 Main St, Boston, MA
02101\nPV1|1|I",
      "validation": {
        "is_valid": true,
        "missing_fields": [],
        "message": "Validation passed"
      },
      "status": "success",
      "mirth_sent": true,
      "mirth_ack": "MSA|AA|abc-123"
    },
    {
      "row_number": 3,
      "patient_id": "PAT002",
      "patient_name": "Jane Smith",
      "hl7_message": "MSH|...",
      "validation": {
        "is_valid": false,
        "missing_fields": ["PID-7"],
        "message": "Missing critical fields"
      },
      "status": "validation_failed"
    }
  ],
  "error": null
}
```

Response Fields:

- `upload_id` (string) - Session ID
- `filename` (string) - Original filename
- `status` (string) - "completed" or "error"
- `current_step` (integer) - Always 6 if completed
- `total_patients` (integer) - Total patient count
- `successful` (integer) - Successfully generated messages
- `failed` (integer) - Failed to generate
- `mirth_successful` (integer) - Successfully sent to Mirth
- `mirth_failed` (integer) - Failed to send to Mirth
- `created_at` (string) - Upload timestamp
- `completed_at` (string) - Completion timestamp
- `messages` (array) - Array of all message results
- `error` (string | null) - Error message if status is "error"

Message Object Fields:

- `row_number` (integer) - Excel row number (starts at 2)
- `patient_id` (string) - Patient ID from CSV

- `patient_name` (string) - Full name
- `hl7_message` (string) - Generated HL7 message
- `validation` (object) - Validation result
- `status` (string) - "success", "validation_failed", or "error"
- `mirth_sent` (boolean, optional) - If sent to Mirth
- `mirth_ack` (string, optional) - ACK from Mirth
- `error` (string, optional) - Error message if generation failed

Use Case:

```
// Frontend: Display results in Step 6 completion screen
async function showResults(uploadId) {
  const results = await fetch(`/api/upload/${uploadId}/results`)
    .then(r => r.json());

  // Show summary
  document.getElementById('total').textContent = results.total_patients;
  document.getElementById('successful').textContent = results.successful;
  document.getElementById('failed').textContent = results.failed;
  document.getElementById('mirth-sent').textContent = results.mirth_successful;

  // Show message table
  const tbody = document.getElementById('results-table');
  results.messages.forEach(msg => {
    const row = tbody.insertRow();
    row.innerHTML = `
      <td>${msg.patient_name}</td>
      <td>${msg.patient_id}</td>
      <td>${msg.status === 'success' ? '✓' : '✗'}</td>
      <td>${msg.mirth_sent ? 'Yes' : 'No'}</td>
      <td><button onclick="showHL7('${escape(msg.hl7_message)}')">View</button></td>
    `;
  });
}
```

Code Structure Explained

File Organization

```
main_with_fastapi.py (1,391 lines)
■
■■■ Imports (Lines 1-36)
■   ■■■ Standard library (os, sys, re, socket, time, etc.)
■   ■■■ Third-party (FastAPI, Pydantic, Pandas, etc.)
■   ■■■ OpenAI SDK (optional)
■
■■■ Configuration (Lines 45-65)
■   ■■■ OPENAI_API_KEY
■   ■■■ MIRTH_HOST, MIRTH_PORT
■   ■■■ FIELD_TIERS (validation)
■   ■■■ TRIGGER_EVENT_MAPPING
```

```
■
■■■ Pydantic Models (Lines 67-96)
■   ■■■ HL7GenerationRequest
■   ■■■ HL7ValidationResponse
■   ■■■ HL7GenerationResponse
■   ■■■ MirthSendResponse
■   ■■■ BatchProcessingResponse
■
■■■ FastAPI App Initialization (Lines 98-146)
■   ■■■ FastAPI app with metadata
■   ■■■ Global client_wrapper
■   ■■■ upload_sessions dictionary
■   ■■■ dashboard_stats dictionary
■
■■■ Helper Functions (Lines 148-172)
■   ■■■ _normalize_colname() - Column name normalization
■   ■■■ _find_column() - Flexible column matching
■
■■■ Client Wrapper Class (Lines 174-222)
■   ■■■ __init__() - Initialize OpenAI client
■   ■■■ generate() - Main generation method
■   ■■■ generate_via_api() - OpenAI API call
■
■■■ HL7 Generation (Lines 224-293)
■   ■■■ fallback_hl7_generator() - Local generator
■   ■■■ generate_hl7_message() - Main entry point
■
■■■ HL7 Validation (Lines 295-337)
■   ■■■ validate_hl7_structure() - Basic parsing
■   ■■■ validate_required_fields_api() - 3-tier validation
■
■■■ Mirth Integration (Lines 339-375)
■   ■■■ send_to_mirth() - MLLP protocol sender
■
■■■ Batch Processing (Lines 377-465)
■   ■■■ process_excel_batch() - Synchronous batch processing
■
■■■ Async Background Processing (Lines 467-675)
■   ■■■ process_csv_with_progress() - 6-step async workflow
■
■■■ API Endpoints (Lines 677-1180)
■   ■■■ startup_event() - Initialize on startup
■   ■■■ root() - API info
■   ■■■ health_check() - Health endpoint
■   ■■■ generate_hl7_from_command() - Text to HL7
■   ■■■ validate_hl7_message() - Validate HL7
■   ■■■ send_hl7_to_mirth() - Send single message
■   ■■■ upload_and_process_excel() - Legacy bulk upload
■   ■■■ get_supported_trigger_events() - Reference data
■   ■■■ get_dashboard_stats() - Dashboard statistics ■ NEW
■   ■■■ get_system_status() - System health ■ NEW
■   ■■■ upload_csv_with_realtime_progress() - Real-time upload ■ NEW
■   ■■■ stream_upload_progress() - SSE stream ■ NEW
■   ■■■ get_upload_status() - Status polling ■ NEW
■   ■■■ get_upload_results() - Final results ■ NEW
■
■■■ Console Mode Functions (Lines 1182-1364)
■   ■■■ upload_excel_file() - File dialog
■   ■■■ validate_required_fields() - Console validation
■   ■■■ display_hl7_details() - Display segments
■   ■■■ console_main() - Main console loop
■
■■■ Main Entry Point (Lines 1366-1391)
    ■■■ main() - Argument parsing and mode selection
```

Key Functions Explained

1. `process_csv_with_progress()` (Lines 467-675)

Purpose: Background async task that processes CSV through 6 steps

Parameters:

- `upload_id` (str) - Session ID to update
- `df` (DataFrame) - Parsed CSV data
- `trigger_event` (str) - HL7 trigger event (default: "ADT-A01")
- `send`*`tomirth`*`_flag` (bool) - Whether to send to Mirth (default: False)

Flow:

```
async def process_csv_with_progress(...):
    session = upload_sessions[upload_id]  # Get session reference

    try:
        # Step 1: File uploaded (0.5s)
        session["current_step"] = 1
        session["step_status"] = "File uploaded successfully"
        await asyncio.sleep(0.5)

        # Step 2: Parse CSV (0.3s + parsing time)
        # - Map column names flexibly
        # - Extract patient data
        # - Normalize dates, addresses
        # - Store in parsed_patients array

        # Step 3: Select patients (0.5s)
        # - Auto-select all patients
        # - Store selected_patient_ids

        # Step 4: Generate HL7 messages
        # - Loop through each patient
        # - Call generate_hl7_message()
        # - Validate generated message
        # - Update progress every iteration
        # - 1 second delay between messages (rate limiting)

        # Step 5: Send to Mirth (if enabled)
        # - Loop through generated messages
        # - Send via MLLP protocol
        # - Track success/failure counts
        # - 0.2 second delay between sends

        # Step 6: Complete
        # - Mark status as "completed"
        # - Set completion timestamp
        # - Update dashboard statistics

    except Exception as e:
        session["status"] = "error"
        session["error"] = str(e)
```

Why Async?

- Non-blocking: Server can handle other requests while processing
- Real-time updates: Can update session state while running

- Rate limiting: Uses `await asyncio.sleep()` for delays
- Background execution: Started with `asyncio.create_task()`

2. `send_to_mirth()` (Lines 339-375)

Purpose: Send HL7 message to Mirth Connect via MLLP protocol

MLLP Protocol:

```
MLLP (Minimal Lower Layer Protocol) Envelope:

<VT> + HL7_MESSAGE + <FS> + <CR>

Where:
  <VT>  = 0x0B (ASCII 11) - Start Block
  <FS>  = 0x1C (ASCII 28) - End Block
  <CR>  = 0x0D (ASCII 13) - Carriage Return
```

Implementation:

```
def send_to_mirth(hl7_message: str, host: str = "localhost", port: int = 6661):
    sock = None
    try:
        # Build MLLP envelope
        START_BLOCK = b"\x0b"
        END_BLOCK = b"\x1c\x0d"
        hl7_bytes = hl7_message.replace("\n", "\r").encode("utf-8")
        mllp_message = START_BLOCK + hl7_bytes + END_BLOCK

        # Create TCP socket
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(15)  # 15 second timeout
        sock.connect((host, port))

        # Send message
        sock.sendall(mllp_message)

        # Wait for ACK/NACK
        response = sock.recv(4096)
        ack_message = response.decode("utf-8", errors="ignore").strip("\x0b\x1c\x0d")

        # Check ACK code
        if "AA" in ack_message or "CA" in ack_message:
            return True, ack_message  # Success
        else:
            return False, ack_message  # Rejected

    except socket.timeout:
        return False, "Connection timeout!"
    except ConnectionRefusedError:
        return False, "Connection refused!"
    finally:
        if sock:
            sock.close()  # Always close socket
```

ACK Codes:

- `AA` - Application Accept (success)

- CA - Commit Accept (success)
- AE - Application Error (rejected)
- AR - Application Reject (rejected)

3. `generate_hl7_message()` (Lines 279-293)

Purpose: Generate HL7 message from patient data

Flow:

```
def generate_hl7_message(client: ClientWrapper, command: str) -> str:
    # Try OpenAI first
    if client.has_remote:
        try:
            return client.generate_via_api(prompt)
        except Exception:
            # Fallback to deterministic generator
            return fallback_hl7_generator(command)
    else:
        # No OpenAI, use fallback
        return fallback_hl7_generator(command)
```

OpenAI Prompt:

```
prompt = f"""You are an HL7 v2.x message generator. Generate a valid HL7 message based on
the following command:

{command}

CRITICAL REQUIREMENTS:
- Use HL7 v2.x pipe-delimited format.
- Auto-populate MSH-11='P', MSH-12='2.5', EVN-2='{current_time}'
- Only include segments explicitly required or implied.
- For ADTs include MSH, EVN, PID, PV1 minimum.
- Output ONLY the HL7 message text (no explanation).
"""
```

Fallback Generator:

- Parses command text using regex
- Extracts patient fields (ID, name, DOB, gender, address)
- Builds HL7 message deterministically
- Always generates valid HL7 v2.5 format

Data Flow & Processing

Complete CSV Processing Flow

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  1. CSV File Input                                                 ■
■  patients.csv:                                                     ■
■  Patient ID,First Name,Last Name,DOB,Gender,Address 1,City        ■
■  PATO01,John,Doe,1990-01-15,M,123 Main St,Boston                 ■
■  PAT002,Jane,Smith,1985-05-20,F,456 Oak Ave,Cambridge            ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                              ■
                              ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  2. Frontend Uploads File                                          ■
■  POST /api/upload                                                  ■
■  - file: patients.csv                                              ■
■  - trigger_event: "ADT-A01"                                        ■
■  - send_to_mirth: true                                             ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                              ■
                              ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  3. Backend Receives Request                                       ■
■  upload_csv_with_realtime_progress():                             ■
■    - Read file contents (BytesIO)                                 ■
■    - Parse CSV → DataFrame (pandas)                                ■
■    - Generate upload_id (UUID)                                    ■
■    - Create session in upload_sessions{}                          ■
■    - Start background task                                        ■
■    - Return upload_id immediately                                 ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                              ■
                              ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  4. Background Processing Starts                                   ■
■  asyncio.create_task(process_csv_with_progress(...))              ■
■                                                                     ■
■  Runs independently in background                                  ■
■  Updates session dictionary as it progresses                       ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                              ■
                              ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  5. Frontend Connects to SSE Stream                                ■
■  GET /api/upload/{upload_id}/stream                               ■
■  EventSource receives updates every 500ms                          ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                              ■
                              ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  6. Step-by-Step Processing (Background)                           ■
■                                                                     ■
■  Step 1 (0.5s):                                                    ■
■    session["current_step"] = 1                                    ■
■    session["step_status"] = "File uploaded successfully"         ■
■                                                                     ■
■  Step 2 (0.5-1s):                                                  ■
■    - Map column names (flexible matching)                         ■
■    - Loop through DataFrame rows                                  ■
■    - Extract: first_name, last_name, dob, gender, address        ■
■    - Normalize DOB (YYYYMMDD format)                              ■
■    - Build parsed_patients array                                  ■
■    session["total_patients"] = 2                                  ■
■                                                                     ■
■  Step 3 (0.5s):                                                    ■
■    - Auto-select all patients                                     ■
■    session["selected_patient_ids"] = ["PAT001", "PAT002"]        ■
■                                                                     ■
■  Step 4 (2-4s per patient):                                        ■
■    For each patient:                                               ■
■      - Build command text                                          ■
■      - Call generate_hl7_message()                              ■
■        ■■ Try OpenAI API (if available)                          ■
■        ■■ Fallback to deterministic generator                    ■
■      - Validate generated message                                 ■
■      - Store in generated_messages array                        ■
■      - Update: "Generated 1/2 messages"                         ■
■      - Wait 1 second (rate limiting)                            ■
■                                                                   ■
■  Step 5 (0.2s per patient, if enabled):                          ■
■    For each successful message:                                  ■
■      - Build MLLP envelope                                       ■
■      - Connect to Mirth (TCP socket)                            ■
■      - Send message                                              ■
■      - Wait for ACK/NACK                                         ■
■      - Track success/failure                                     ■
■      - Wait 0.2 seconds                                          ■
■                                                                   ■
■  Step 6 (immediate):                                             ■
■    session["status"] = "completed"                              ■
■    session["completed_at"] = "2025-12-20T14:00:15"             ■
■    dashboard_stats["total_processed"] += 2                      ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                              ■
                              ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  7. SSE Stream Sends Final Event                                 ■
■  data: {"current_step":6,"status":"completed",...}             ■
■  Frontend receives completion event                              ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                              ■
                              ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  8. Frontend Loads Results                                       ■
■  GET /api/upload/{upload_id}/results                            ■
■  Displays:                                                       ■
■    - Total: 2 patients                                           ■
■    - Successful: 2 messages                                      ■
■    - Sent to Mirth: 2                                            ■
■    - Table with all messages                                     ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

Flexible CSV Column Mapping

Problem: CSV files have different column naming conventions

Examples:

- "First Name" vs "FirstName" vs "first_name" vs "Given Name"
- "DOB" vs "Date of Birth" vs "birthdate"
- "Gender" vs "Sex"

Solution: Flexible column mapping with normalization

```
# Define multiple possible names for each field
candidate_map = {
    'patient_first_name': [
        "Patient First Name",
        "First Name",
        "Firstname",
        "given_name",
        "first_name"
    ],
    'dob': [
        "DOB",
        "Date of Birth",
        "birthdate",
        "date_of_birth"
    ],
    'gender': [
        "Gender",
        "Sex"
    ]
}

# Normalize column names for comparison
def _normalize_colname(name: str) -> str:
    s = name.strip().lower()
    s = re.sub(r"[^\w\s]", " ", s)  # Remove punctuation
    s = re.sub(r"\s+", " ", s)      # Multiple spaces → single space
    s = s.replace(" ", "_")           # Spaces → underscores
    return s

# Find best matching column
for key, candidates in candidate_map.items():
    found = _find_column(df, candidates)
    mapping[key] = found

# Extract patient data using mapped columns
first_name = row.get(mapping.get('patient_first_name'))
```

Result: Works with ANY reasonable CSV column naming!

Real-Time Features

Server-Sent Events (SSE) Explained

What is SSE?

Server-Sent Events (SSE) is a standard for real-time server-to-client communication:

Characteristics:

- **One-way:** Server pushes updates to client
- **HTTP:** Uses standard HTTP (not WebSocket)
- **Text-based:** Sends text data
- **Auto-reconnect:** Browser automatically reconnects if connection drops

- **Simple:** Easier to implement than WebSockets

SSE vs Polling vs WebSockets

Feature	SSE	Polling	WebSockets
Direction	Server → Client	Client → Server	Bidirectional
Protocol	HTTP	HTTP	Custom protocol
Complexity	Low	Very low	High
Reconnection	Automatic	Manual	Manual
Server load	Low	High	Low
Browser support	All modern browsers	All browsers	All modern browsers
Use case	Progress updates, live feeds	Simple status checks	Chat, gaming, collaboration

When to use what:

- **SSE:** Progress updates, notifications, live data feeds (our use case)
- **Polling:** Simple status checks, older browser support required
- **WebSockets:** Real-time bidirectional communication (chat, gaming)

SSE Implementation in Code

Backend (FastAPI):

```
@app.get("/api/upload/{upload_id}/stream")
async def stream_upload_progress(upload_id: str):
    async def event_generator():
        session = upload_sessions[upload_id]

        # Stream events while processing
        while session["status"] == "processing":
            # Build event data
            event_data = {
                "current_step": session.get("current_step", 0),
                "step_status": session.get("step_status", ""),
                "status": session["status"]
            }

            # Send SSE event (CRITICAL FORMAT!)
            yield f"data: {json.dumps(event_data)}\n\n"
            #     ^^^^                        ^^^^
            #     Must start with "data:"    Two newlines required!

            # Wait before next update
            await asyncio.sleep(0.5)

        # Send final event
        final_data = {"status": "completed", ...}
        yield f"data: {json.dumps(final_data)}\n\n"

    # Return streaming response
    return StreamingResponse(
        event_generator(),
        media_type="text/event-stream"  # CRITICAL!
    )
```

Frontend (JavaScript):

```
// Create EventSource connection
const eventSource = new EventSource('/api/upload/abc-123/stream');

// Handle incoming events
eventSource.onmessage = (event) => {
  const data = JSON.parse(event.data);
  console.log('Received:', data);

  // Update UI based on data
  updateUI(data);

  // Close when done
  if (data.status === 'completed') {
    eventSource.close();
  }
};

// Handle errors
eventSource.onerror = (error) => {
  console.error('SSE error:', error);
  eventSource.close();
};
```

SSE Message Format

HTTP Response:

```
HTTP/1.1 200 OK
Content-Type: text/event-stream
Cache-Control: no-cache
Connection: keep-alive

data: {"current_step":1,"step_status":"File uploaded"}\n\n

data: {"current_step":2,"step_status":"Parsing CSV..."}\n\n

data: {"current_step":3,"step_status":"Selected 10 patients"}\n\n
```

Critical Requirements:

1. Content-Type MUST be `text/event-stream`
2. Each event MUST start with `data:`
3. Each event MUST end with `\n\n` (two newlines)
4. Connection MUST stay open (keep-alive)

Integration Guide

Complete Frontend Integration Example

Step 1: Upload CSV File

```
// HTML
<input type="file" id="csvFile" accept=".csv,.xlsx">
<select id="triggerEvent">
  <option value="ADT-A01">ADT-A01 - Register Patient</option>
  <option value="ADT-A04">ADT-A04 - Register Outpatient</option>
</select>
<label>
  <input type="checkbox" id="sendToMirth"> Send to Mirth
</label>
<button onclick="uploadFile()">Upload</button>

// JavaScript
async function uploadFile() {
  const fileInput = document.getElementById('csvFile');
  const triggerEvent = document.getElementById('triggerEvent').value;
  const sendToMirth = document.getElementById('sendToMirth').checked;

  if (!fileInput.files.length) {
    alert('Please select a file');
    return;
  }

  const formData = new FormData();
  formData.append('file', fileInput.files[0]);
  formData.append('trigger_event', triggerEvent);
  formData.append('send_to_mirth', sendToMirth);

  try {
    const response = await fetch('http://localhost:8000/api/upload', {
      method: 'POST',
      body: formData
    });

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }

    const data = await response.json();
    console.log('Upload started:', data.upload_id);

    // Start real-time progress tracking
    connectToProgressStream(data.upload_id);

  } catch (error) {
    console.error('Upload failed:', error);
    alert('Upload failed: ' + error.message);
  }
}
```

Step 2: Connect to Progress Stream

```
// HTML
<div id="wizardSteps">
  <div class="step" data-step="1">1. Upload CSV</div>
  <div class="step" data-step="2">2. Parse Data</div>
  <div class="step" data-step="3">3. Select Patients</div>
  <div class="step" data-step="4">4. Create HL7</div>
  <div class="step" data-step="5">5. Push to EMR</div>
  <div class="step" data-step="6">6. Complete</div>
</div>
<div id="statusMessage"></div>
<div id="progressBar">
  <div id="progressFill" style="width: 0%"></div>
  <span id="progressText">0/0</span>
</div>

// JavaScript
function connectToProgressStream(uploadId) {
  const eventSource = new EventSource(`http://localhost:8000/api/upload/${uploadId}/
stream`);

  eventSource.onmessage = (event) => {
    const data = JSON.parse(event.data);
    console.log('Progress update:', data);

    // Update wizard steps
    highlightStep(data.current_step);

    // Update status message
    document.getElementById('statusMessage').textContent = data.step_status;

    // Update progress bar (if in step 4)
    if (data.generated_count && data.total_patients) {
      const percent = (data.generated_count / data.total_patients) * 100;
      document.getElementById('progressFill').style.width = percent + '%';
      document.getElementById('progressText').textContent =
        `${data.generated_count}/${data.total_patients}`;
    }

    // When complete, close connection and load results
    if (data.status === 'completed') {
      eventSource.close();
      console.log('Processing complete!');
      loadResults(uploadId);
    }

    // Handle errors
    if (data.status === 'error') {
      eventSource.close();
      alert('Error: ' + data.error);
    }
  };

  eventSource.onerror = (error) => {
    console.error('SSE connection error:', error);
    eventSource.close();
    alert('Connection lost. Please refresh and try again.');
  };
}

function highlightStep(stepNumber) {
  // Remove active class from all steps
  document.querySelectorAll('.step').forEach(el => {
    el.classList.remove('active');
  });

  // Add active class to current step
  const currentStep = document.querySelector(`[data-step="${stepNumber}"]`);
  if (currentStep) {
    currentStep.classList.add('active');
  }
}
```

Step 3: Load Final Results

```
// HTML
<div id="resultsScreen" style="display:none">
  <h2>Processing Complete!</h2>
  <div class="summary">
    <div>Total Patients: <span id="totalPatients"></span></div>
    <div>Successful: <span id="successfulCount"></span></div>
    <div>Failed: <span id="failedCount"></span></div>
    <div>Sent to Mirth: <span id="mirthSentCount"></span></div>
  </div>
  <table id="resultsTable">
    <thead>
      <tr>
        <th>Patient Name</th>
        <th>Patient ID</th>
        <th>Status</th>
        <th>Sent to Mirth</th>
        <th>Actions</th>
      </tr>
    </thead>
    <tbody id="resultsTableBody"></tbody>
  </table>
</div>

// JavaScript
async function loadResults(uploadId) {
  try {
    const response = await fetch(`http://localhost:8000/api/upload/${uploadId}/results`);
    const results = await response.json();

    // Show results screen
    document.getElementById('resultsScreen').style.display = 'block';

    // Update summary
    document.getElementById('totalPatients').textContent = results.total_patients;
    document.getElementById('successfulCount').textContent = results.successful;
    document.getElementById('failedCount').textContent = results.failed;
    document.getElementById('mirthSentCount').textContent = results.mirth_successful;

    // Populate results table
    const tbody = document.getElementById('resultsTableBody');
    tbody.innerHTML = '';  // Clear existing rows

    results.messages.forEach(msg => {
      const row = tbody.insertRow();

      // Patient name
      const nameCell = row.insertCell();
      nameCell.textContent = msg.patient_name;

      // Patient ID
      const idCell = row.insertCell();
      idCell.textContent = msg.patient_id;

      // Status
      const statusCell = row.insertCell();
      statusCell.textContent = msg.status === 'success' ? '✓ Success' : '✗ Failed';
      statusCell.className = msg.status === 'success' ? 'success' : 'error';

      // Sent to Mirth
      const mirthCell = row.insertCell();
      mirthCell.textContent = msg.mirth_sent ? 'Yes' : 'No';

      // Actions
      const actionsCell = row.insertCell();
      const viewButton = document.createElement('button');
      viewButton.textContent = 'View HL7';
      viewButton.onclick = () => showHL7Message(msg.hl7_message);
      actionsCell.appendChild(viewButton);
    });

  } catch (error) {
    console.error('Failed to load results:', error);
    alert('Failed to load results: ' + error.message);
  }
}

function showHL7Message(hl7Message) {
  // Show HL7 message in modal or popup
  const modal = document.createElement('div');
  modal.className = 'modal';
  modal.innerHTML = `
    <div class="modal-content">
      <h3>HL7 Message</h3>
      <pre>${hl7Message}</pre>
      <button onclick="this.parentElement.parentElement.remove()">Close</button>
    </div>
  `;
  document.body.appendChild(modal);
}
```

Step 4: Update Dashboard

```
// HTML
<div id="dashboard">
  <div class="tile">
    <h3>Total Processed</h3>
    <div id="totalProcessed" class="count">0</div>
  </div>
  <div class="tile">
    <h3>HL7 Messages</h3>
    <div id="hl7Messages" class="count">0</div>
  </div>
  <div class="tile">
    <h3>Success Rate</h3>
    <div id="successRate" class="count">0%</div>
  </div>
  <div class="system-status">
    <div>
      Mirth: <span id="mirthStatus" class="status-dot"></span>
    </div>
    <div>
      HL7 Parser: <span id="parserStatus" class="status-dot"></span>
    </div>
  </div>
</div>

// JavaScript
// Poll dashboard stats every 5 seconds
setInterval(async () => {
  try {
    const stats = await fetch('http://localhost:8000/api/dashboard/stats')
      .then(r => r.json());

    document.getElementById('totalProcessed').textContent = stats.total_processed;
    document.getElementById('hl7Messages').textContent = stats.hl7_messages;
    document.getElementById('successRate').textContent = stats.success_rate + '%';

  } catch (error) {
    console.error('Failed to fetch dashboard stats:', error);
  }
}, 5000);

// Poll system status every 10 seconds
setInterval(async () => {
  try {
    const status = await fetch('http://localhost:8000/api/dashboard/system-status')
      .then(r => r.json());

    // Update Mirth status indicator
    const mirthDot = document.getElementById('mirthStatus');
    if (status.openemr_connection.status === 'Active') {
      mirthDot.className = 'status-dot green';
```

```
        mirthDot.title = 'Mirth Connected';
      } else {
        mirthDot.className = 'status-dot red';
        mirthDot.title = 'Mirth Offline';
      }

      // Update parser status
      const parserDot = document.getElementById('parserStatus');
      if (status.hl7_parser.status === 'Running') {
        parserDot.className = 'status-dot green';
        parserDot.title = 'OpenAI Enabled';
      } else {
        parserDot.className = 'status-dot yellow';
        parserDot.title = 'Using Fallback Generator';
      }

    } catch (error) {
      console.error('Failed to fetch system status:', error);
    }
  }, 10000);
```

Testing Guide

Testing in Swagger UI

Step 1: Start the Server

```
cd /Users/nagarajm/Work/SG/interface-wizard/actual-code
python main_with_fastapi.py --api
```

Step 2: Open Swagger UI

Navigate to: **http://localhost:8000/docs**

Step 3: Test Dashboard Stats

1. Find **Dashboard** section
2. Click on `GET /api/dashboard/stats`
3. Click "Try it out"
4. Click "Execute"
5. See response (should show all zeros initially)

Step 4: Test System Status

1. Find `GET /api/dashboard/system-status`
2. Click "Try it out"
3. Click "Execute"
4. Check `openemr_connection.status` - should show "Active" if Mirth is running

Step 5: Test Real-Time Upload

1. Find **Real-Time Upload** section
2. Click on `POST /api/upload`
3. Click "Try it out"
4. Click "Choose File" and select `test_patients.csv`
5. Set `trigger_event` = "ADT-A01"
6. Set `send`*`to`*`mirth` = true
7. Click "Execute"
8. Copy the `upload_id` from response

Step 6: Test SSE Stream (Can't test in Swagger - use cURL)

```
# Replace with your actual upload_id
curl -N "http://localhost:8000/api/upload/abc-123-def-456/stream"
```

You'll see real-time events stream in your terminal!

Step 7: Test Final Results

1. Wait for processing to complete (~15 seconds for 5 patients)
2. Find `GET /api/upload/{upload_id}/results`
3. Click "Try it out"
4. Enter your upload_id
5. Click "Execute"
6. See complete results with all messages

Testing with cURL

Test Dashboard Stats:

```
curl http://localhost:8000/api/dashboard/stats
```

Test Upload:

```
curl -X POST "http://localhost:8000/api/upload" \
  -F "file=@test_patients.csv" \
  -F "trigger_event=ADT-A01" \
  -F "send_to_mirth=true"
```

Test SSE Stream:

```
# Use -N flag for no buffering
curl -N "http://localhost:8000/api/upload/{upload_id}/stream"
```

Test Results:

```
curl "http://localhost:8000/api/upload/{upload_id}/results"
```

Troubleshooting

Common Issues

1. "Connection refused" on Mirth

Symptom:

```
Connection refused! Check Mirth is running and channel is listening.
```

Causes:

- Mirth Connect not running
- Channel not deployed/started
- Wrong port (should be 6661)

Solution:

1. Open Mirth Connect Administrator (https://localhost:8443)
2. Go to Channels tab
3. Verify "IW Listener HL7" channel exists
4. Check status is "Started" (green)
5. Verify listener port is 6661

Verify Connection:

```
# Test if port 6661 is listening
netstat -an | grep 6661

# Or telnet test
telnet localhost 6661
```

---

2. "Upload session not found"

Symptom:

```
{
  "detail": "Upload session not found"
}
```

Causes:

- Invalid upload_id
- Server restarted (sessions lost)
- Session expired (unlikely with in-memory)

Solution:

1. Verify you copied the correct upload_id
2. Check if server restarted (sessions are in-memory)
3. Upload file again to get new upload_id

---

3. Dashboard stats show 0

Symptom:

All dashboard stats are 0

Causes:

- No uploads processed yet
- Server restarted (stats reset)

Solution:

This is normal! Stats will update after processing uploads.

---

4. SSE stream not connecting

Symptom:

EventSource connection fails or times out

Causes:

- CORS issues
- Invalid upload_id
- Browser doesn't support SSE (rare)

Solution:

Check CORS:

```
# In main_with_fastapi.py, add CORS middleware if needed
from fastapi.middleware.cors import CORSMiddleware

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # Or specific origins
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Use polling instead:

```
// If SSE doesn't work, use polling
```

```
async function pollStatus(uploadId) {
  const interval = setInterval(async () => {
    const status = await fetch(`/api/upload/${uploadId}/status`)
      .then(r => r.json());

    if (status.status === 'completed') {
      clearInterval(interval);
    }
  }, 1000);
}
```

5. OpenAI API errors

Symptom:

```
■ Remote generation failed: ... Falling back to local generator.
```

Causes:

- No API key configured
- API key invalid
- Rate limit exceeded
- Network error

Solution:

Check API key:

```
# In main_with_fastapi.py, line 46
OPENAI_API_KEY = "sk-..."  # Replace with valid key
```

Fallback works fine:

The system automatically uses the deterministic fallback generator if OpenAI fails. Messages will still be valid HL7 v2.5 format!

Glossary

Healthcare/HL7 Terms

HL7 (Health Level 7)

- International standard for healthcare data exchange
- Version 2.x is pipe-delimited text format
- Version 3.x and FHIR are XML/JSON based

ADT (Admission/Discharge/Transfer)

- Message type for patient movement
- ADT-A01 = Admit/Register Patient
- ADT-A04 = Register Outpatient
- ADT-A08 = Update Patient Information

MLLP (Minimal Lower Layer Protocol)

- Protocol for sending HL7 messages over TCP/IP
- Wraps message with special bytes:
- Start: (0x0B)
- End: (0x1C 0x0D)

Mirth Connect

- Open-source HL7 message router
- Receives, transforms, and routes HL7 messages
- Has channels with source/destination connectors

OpenEMR

- Open-source Electronic Medical Record system
- Uses MySQL database
- Stores patient data in `patient_data` table

ACK (Acknowledgment)

- Response message confirming receipt
- AA = Application Accept (success)
- AE = Application Error (rejected)

Technical Terms

FastAPI

- Modern Python web framework
- Built on Starlette and Pydantic
- Automatic API documentation (Swagger/OpenAPI)
- Async/await support

Pydantic

- Data validation using Python type hints
- Automatic JSON serialization
- Runtime type checking

Server-Sent Events (SSE)

- Standard for server-to-client real-time updates
- Uses HTTP (not WebSocket)
- One-way communication
- Auto-reconnect

Async/Await

- Python concurrency feature
- Non-blocking I/O operations
- Allows multiple tasks to run concurrently

UUID (Universally Unique Identifier)

- 128-bit unique identifier
- Example: `abc-123-def-456`
- Used for session IDs

In-Memory Storage

- Data stored in RAM (Python dictionaries)
- Fast access
- Lost on restart
- Alternative: Database persistence

Rate Limiting

- Intentional delay between operations
- Prevents overwhelming services
- Example: 1 second between HL7 generations

CORS (Cross-Origin Resource Sharing)

- Security feature for web browsers
- Controls which domains can access API
- Needed for frontend-backend communication

Appendix: Example HL7 Message

```
MSH|^~\&|SMART_APP|SMART_FAC|REC_APP|REC_FAC|20251220140030||ADT^A01|abc-123-def-456|P|2.5
EVN|A01|20251220140030
PID|1||PAT001||Doe^John||19900115|M|||123 Main St, Boston, MA 02101
PV1|1|I
```

Segment Breakdown:

MSH (Message Header):

- Field 1: `MSH` - Segment type
- Field 2: `^~\&` - Encoding characters
- Field 3: `SMART_APP` - Sending application
- Field 4: `SMART_FAC` - Sending facility
- Field 5: `REC_APP` - Receiving application
- Field 6: `REC_FAC` - Receiving facility
- Field 7: `20251220140030` - Timestamp (YYYYMMDDHHmmss)
- Field 9: `ADT^A01` - Message type (ADT) and trigger event (A01)
- Field 10: `abc-123-def-456` - Message control ID (unique)
- Field 11: `P` - Processing ID (P = Production)
- Field 12: `2.5` - HL7 version

EVN (Event):

- Field 1: `A01` - Event type code
- Field 2: `20251220140030` - Event timestamp

PID (Patient Identification):

- Field 1: `1` - Set ID
- Field 3: `PAT001` - Patient identifier (MRN)
- Field 5: `Doe^John` - Patient name (Last^First)
- Field 7: `19900115` - Date of birth (YYYYMMDD)
- Field 8: `M` - Gender (M/F/U)
- Field 11: `123 Main St, Boston, MA 02101` - Address

PV1 (Patient Visit):

- Field 1: `1` - Set ID
- Field 2: `I` - Patient class (I = Inpatient, O = Outpatient)

Conclusion

This documentation provides a complete guide to the Interface Wizard Backend API. Key takeaways:

■ **6 new endpoints** for real-time CSV processing and dashboard statistics

■ **Server-Sent Events (SSE)** for live progress updates

■ **In-memory storage** for fast, simple session management

■ **Mirth Connect integration** via MLLP protocol

■ **OpenAI-powered** HL7 generation with fallback

■ **Complete Swagger documentation** at `/docs`

■ **Production-ready** with proper error handling and validation

Next Steps

For Backend Developers:

1. Review code structure in `mainwithfastapi.py`
2. Test endpoints in Swagger UI
3. Understand SSE implementation
4. Customize as needed for your environment

For Frontend Developers:

1. Review Integration Guide section
2. Test API endpoints from your framework
3. Implement SSE connection
4. Build 6-step wizard UI

For System Administrators:

1. Deploy FastAPI application
2. Configure Mirth Connect
3. Set up OpenAI API key
4. Monitor logs and performance

For QA Engineers:

1. Follow Testing Guide
2. Test with sample CSV files
3. Verify Mirth integration
4. Test error scenarios

Document Version: 1.0

Last Updated: December 20, 2025

For Questions: Contact Interface Wizard Development Team