Complete API Documentation - main_with_fastapi.py

📚 Table of Contents

🎯 Overview

File: `main_with_fastapi.py`

Purpose: Smart HL7 v2.x message generator with dual operation modes:

- API Mode: REST API with Swagger documentation
- Console Mode: Interactive command-line interface

Technology Stack:

- FastAPI - REST API framework
- Uvicorn - ASGI server
- Pydantic - Data validation
- OpenAI GPT-4 - AI-powered generation
- HL7 library - Message parsing
- Pandas - Excel/CSV processing

🏗️ Architecture

```
┌─────────────────────────────────────────────────────────────┐
│                    main_with_fastapi.py                      │
├─────────────────────────────────────────────────────────────┤
│                                                               │
│  ┌──────────────┐              ┌──────────────┐             │
│  │  API Mode    │              │ Console Mode │             │
│  │  (FastAPI)   │              │  (Original)  │             │
│  └──────┬───────┘              └──────┬───────┘             │
│         │                               │                     │
│         └───────────┬───────────────────┘                     │
│                     │                                         │
│         ┌───────────▼──────────┐                             │
│         │   Core Functions      │                             │
│         │  (Shared Logic)      │                             │
│         └───────────┬──────────┘                             │
│                     │                                         │
│    ┌────────────────┼────────────────┐                       │
│    │                │                │                        │
│ ┌──▼───┐    ┌─────▼────┐    ┌─────▼─────┐                │
│ │OpenAI│    │ HL7 Gen  │    │   Mirth    │                 │
│ │ GPT-4│    │Validation│    │  Connect   │                 │
│ └──────┘    └──────────┘    └────────────┘                │
│                                                               │
└─────────────────────────────────────────────────────────────┘
```

🚀 Setup & Installation

Prerequisites

```
Python 3.8+
pip (Python package manager)
```

Install Dependencies

```
cd IW
pip install -r requirements.txt
```

Dependencies:

```
fastapi==0.109.0           # REST API framework
uvicorn[standard]==0.27.0  # ASGI server
pydantic==2.5.3            # Data validation
pandas==2.1.4              # Excel/CSV processing
```

```
openpyxl==3.1.2          # Excel file support
hl7==0.4.5               # HL7 message parsing
openai==1.7.2            # OpenAI API client
python-multipart==0.0.6  # File upload support
```

Start Server

API Mode:

```
python main_with_fastapi.py --api
```

Console Mode:

```
python main_with_fastapi.py --console
# or simply
python main_with_fastapi.py
```

Custom Port:

```
python main_with_fastapi.py --api --port 9000 --host 127.0.0.1
```

📡 API Endpoints

1. Root Endpoint

GET `/`

Purpose: API information and health check

Logic:

1. Returns basic API metadata
2. Lists available features
3. Provides links to documentation

Request:

```
curl http://localhost:8000/
```

Response:

```
{
  "name": "Smart HL7 Message Generator API",
  "version": "2.0.0",
  "description": "Generate, validate, and send HL7 v2.x messages",
  "documentation": "/docs",
  "health": "/health",
  "features": [
    "HL7 message generation from text",
    "Bulk processing from Excel/CSV",
    "HL7 message validation",
    "Mirth Connect integration",
    "AI-powered generation (OpenAI GPT-4)"
  ]
}
```

Code Location: Lines 453-469

---

2. Health Check

GET `/health`

Purpose: Check API status and configuration

Logic:

1. Returns current timestamp
2. Shows OpenAI availability status
3. Displays Mirth Connect configuration
4. Indicates server health

Request:

```
curl http://localhost:8000/health
```

Response:

```
{
  "status": "healthy",
  "timestamp": "2025-12-04T10:30:00.123456",
  "openai_available": true,
  "mirth_config": "localhost:6661"
}
```

Code Location: Lines 471-479

Logic Flow:

```
Request → Check client_wrapper exists
       → Get current timestamp
       → Check OpenAI status
       → Return JSON response
```

---

3. Generate HL7 from Command

POST `/api/generate-hl7`

Purpose: Generate HL7 message from natural language text

Logic:

1. Receives natural language command
2. Builds prompt for AI/fallback generator
3. Calls `generate_hl7_message()` function
4. Validates generated HL7 message
5. Returns HL7 message + validation results

Request Body:

```
{
  "command": "Create ADT-A01 for patient John Doe, ID 12345, DOB 1985-03-15,
Gender M, Address: 123 Main St, Boston, MA",
  "trigger_event": "ADT-A01"
}
```

Response:

```
{
  "hl7_message":
"MSH|^~\\&|SMART_APP|SMART_FAC|REC_APP|REC_FAC|20251204103000||ADT^A01||P|2.5\n
EVN|A01|20251204103000\nPID|1|12345||Doe^John||19850315|M|||123 Main St,
Boston, MA\nPV1|1|I",
  "validation": {
    "is_valid": true,
    "missing_fields": [],
    "message": "Validation passed"
  },
  "patient_info": {
    "trigger_event": "ADT-A01",
    "command": "Create ADT-A01 for patient..."
  }
}
```

Code Location: Lines 481-511

Logic Flow:

```
1. Receive request → Extract command and trigger_event
2. Build AI prompt → Include timestamp, formatting rules
3. Call generate_hl7_message(client_wrapper, command)
   ├─ Try OpenAI API (if available)
   │  ├─ Call GPT-4o-mini model
   │  └─ Parse response, strip markdown
   └─ Fallback to local generator (if OpenAI fails)
      ├─ Extract Patient ID via regex
      ├─ Extract Patient Name via regex
      ├─ Extract DOB, Gender, Address via regex
      └─ Build HL7 segments (MSH, EVN, PID, PV1)
4. Validate HL7 message
   ├─ Parse HL7 structure
   ├─ Check critical fields (PID-3, PID-5, PID-7, PID-8)
   └─ Return validation results
5. Return JSON response
```

Error Handling:

- Invalid command → 500 error with detail
- OpenAI failure → Automatic fallback to local generator
- Validation failure → Returns `is_valid: false` with missing fields

---

4. Validate HL7 Message

POST `/api/validate-hl7`

Purpose: Validate an existing HL7 message

Logic:

1. Receives HL7 message as string
2. Parses HL7 structure
3. Checks critical fields (3-tier validation)
4. Returns validation results

Request (Form Data):

```
hl7_message: MSH|^~\&|SMART_APP|...\nEVN|A01|...\nPID|...
```

Response:

```
{
  "is_valid": true,
  "missing_fields": [],
  "message": "Validation passed"
}
```

Response (with errors):

```
{
  "is_valid": false,
  "missing_fields": ["PID-3", "PID-5", "PID-7"],
  "message": "Missing critical fields"
}
```

Code Location: Lines 513-532

Logic Flow:

```
1. Receive HL7 message string
2. validate_required_fields_api(hl7_message)
   ├─ Parse HL7 structure using hl7.parse()
   ├─ Iterate through segments (MSH, EVN, PID, PV1, etc.)
   ├─ For each critical segment:
   │  ├─ Check MSH-9 (Message Type)
   │  ├─ Check PID-3 (Patient ID)
   │  ├─ Check PID-5 (Patient Name)
   │  ├─ Check PID-7 (Date of Birth)
   │  └─ Check PID-8 (Gender)
   └─ Return validation result
3. Return JSON response
```

3-Tier Validation System:

```
Tier 1: System Fields (Auto-populated)
  - MSH-11 (Processing ID) → Always "P"
  - MSH-12 (Version) → Always "2.5"
  - EVN-2 (Timestamp) → Auto-generated
  → NOT VALIDATED (automatically correct)

Tier 2: Contextual Fields (Smart defaults)
  - PV1-2 (Patient Class) → Inferred from trigger
  - ORC-1 (Order Control) → Smart default
  → NOT VALIDATED (inferred correctly)
```

```
Tier 3: Critical Fields (User data)
  - PID-3 (Patient ID) → VALIDATED
  - PID-5 (Patient Name) → VALIDATED
  - PID-7 (Date of Birth) → VALIDATED
  - PID-8 (Gender) → VALIDATED
  → STRICTLY VALIDATED (must be present)
```

---

5. Send to Mirth Connect

POST `/api/send-to-mirth`

Purpose: Send HL7 message to Mirth Connect via MLLP

Logic:

1. Receives HL7 message
2. Wraps message in MLLP envelope
3. Opens TCP socket to Mirth
4. Sends message via MLLP protocol
5. Receives acknowledgment (ACK)
6. Returns success status

Request (Form Data):

```
hl7_message: MSH|^~\&|SMART_APP|...\nEVN|...\nPID|...
```

Response:

```
{
  "success": true,
  "message": "Message sent successfully",
  "acknowledgment": "MSH|^~\\&|MIRTH|...\nMSA|AA|..."
}
```

Code Location: Lines 534-555

Logic Flow:

```
1. Receive HL7 message
2. send_to_mirth(hl7_message, host, port)
   ├─ Prepare MLLP envelope:
   │  ├─ START_BLOCK = \x0b (ASCII 11)
   │  ├─ HL7 message (with \r line endings)
   │  └─ END_BLOCK = \x1c\x0d (ASCII 28 + 13)
   ├─ Create TCP socket
   ├─ Connect to Mirth (localhost:6661)
   ├─ Send MLLP message
   ├─ Receive ACK response
   │  ├─ Parse ACK message
   │  ├─ Check for AA (Application Accept)
   │  └─ Check for CA (Commit Accept)
   └─ Close socket
3. Return JSON response with ACK
```

MLLP Protocol:

```
┌──────────────────────────────────────┐
│  MLLP Message Structure              │
├──────────────────────────────────────┤
│  \x0b (Start Block)                  │
│  MSH|^~\&|...                        │
│  EVN|...                             │
│  PID|...                             │
│  PV1|...                             │
│  \x1c\x0d (End Block + CR)           │
└──────────────────────────────────────┘
```

Error Handling:

- Connection timeout (10 seconds) → Returns error
- Connection refused → Returns "Mirth not running"
- Socket error → Returns detailed error message

---

6. Bulk Upload Excel/CSV

POST `/api/upload-excel`

Purpose: Upload Excel/CSV and generate HL7 messages for all patients

Logic:

1. Receives file upload (Excel or CSV)
2. Parses file using pandas
3. Identifies columns (flexible naming)
4. Generates HL7 message for each patient
5. Validates all messages
6. Optionally sends to Mirth
7. Returns batch results

Request (Form Data):

```
file: [Excel/CSV file]
trigger_event: "ADT-A01"
send_to_mirth_flag: false
```

Response:

```
{
  "total_patients": 10,
  "successful": 9,
  "failed": 1,
  "messages": [
    {
      "row_number": 2,
      "patient_id": "PAT000001",
      "patient_name": "John Doe",
      "hl7_message": "MSH|^~\\&|...",
      "validation": {
        "is_valid": true,
        "missing_fields": []
      },
      "status": "success"
    },
    {
      "row_number": 3,
      "patient_id": "PAT000002",
      "patient_name": "Jane Smith",
      "error": "Missing Date of Birth",
      "status": "error"
    }
  ]
}
```

Code Location: Lines 557-615

Detailed Logic Flow:

```
1. Receive file upload
   ├─ Check file extension (.xlsx, .xls, .csv)
   └─ Read file content into memory

2. Parse file with pandas
   ├─ If CSV: pd.read_csv(BytesIO(contents))
   └─ If Excel: pd.read_excel(BytesIO(contents))

3. Identify columns (flexible naming)
   ├─ Normalize column names (lowercase, remove spaces)
   ├─ Map to standard fields:
   │  ├─ patient_last_name: ["Last Name", "Lastname", "surname"]
   │  ├─ patient_first_name: ["First Name", "Firstname", "given_name"]
   │  ├─ dob: ["DOB", "Date of Birth", "birthdate"]
   │  ├─ gender: ["Gender", "Sex"]
   │  ├─ address1: ["Address 1", "Street", "address"]
   │  ├─ city: ["City", "Town"]
   │  ├─ state: ["State", "Province"]
   │  ├─ zipcode: ["Zipcode", "Zip", "Postal Code"]
   │  └─ patient_id: ["Patient ID", "MRN", "id"]
   └─ Use best matching column for each field

4. For each row in dataframe:
   ├─ Extract patient data
   │  ├─ Get first_name from mapped column
   │  ├─ Get last_name from mapped column
   │  ├─ Get DOB (normalize to YYYYMMDD format)
   │  ├─ Get gender (M/F/U)
   │  └─ Get address components
   ├─ Build command string:
   │  "Trigger Event: ADT-A01
   │   Patient ID: 12345
   │   Patient Name: Doe John
   │   Date of Birth: 19850315
   │   Gender: M
   │   Address: 123 Main St, Boston, MA 02101"
   ├─ Generate HL7 message
   │  └─ Call generate_hl7_message(client_wrapper, command)
   ├─ Validate HL7 message
   │  └─ Call validate_required_fields_api(hl7_message)
   └─ Add result to results list

5. If send_to_mirth_flag = true:
   ├─ For each successful message:
   │  ├─ Call send_to_mirth(hl7_message)
   │  └─ Add mirth_sent status to result
   └─ Count successful/failed transmissions

6. Calculate summary
   ├─ total_patients = len(results)
   ├─ successful = count(status == "success")
   └─ failed = total - successful

7. Return JSON response with all messages
```

Column Mapping Example:

```
Excel Column        → Normalized     → Standard Field
"Patient Last Name" → patient_last_name → patient_last_name
```

```
"First Name"          → first_name      → patient_first_name
"DOB"                 → dob             → dob
"Gender"              → gender          → gender
"Address 1"           → address_1       → address1
```

Error Handling:

- Invalid file format → 500 error
- Missing critical columns → Generates PAT000001 IDs
- Invalid DOB format → Attempts multiple date parsers
- Missing data → Generates with empty fields (may fail validation)

---

7. Get Supported Trigger Events

GET `/api/supported-events`

Purpose: Get list of supported HL7 trigger events

Logic:

1. Returns predefined list of trigger events
2. Returns patient class mappings
3. Provides event descriptions

Request:

```
curl http://localhost:8000/api/supported-events
```

Response:

```
{
  "trigger_events": {
    "ADT-A01": "Admit/Register Patient (Inpatient)",
    "ADT-A02": "Transfer Patient (Inpatient)",
    "ADT-A03": "Discharge Patient (Inpatient)",
    "ADT-A04": "Register Outpatient",
    "ADT-A05": "Pre-Admit Patient",
    "ADT-A08": "Update Patient Information",
    "ADT-A11": "Cancel Admit",
    "ADT-A13": "Cancel Discharge"
  },
  "patient_class_mapping": {
    "A01": "I",
    "A02": "I",
    "A03": "I",
    "A04": "O",
    "A05": "P",
    "A08": "I",
    "A11": "I",
    "A13": "I"
  }
}
```

Code Location: Lines 617-638

Patient Class Mapping:

- I = Inpatient
- O = Outpatient
- P = Pre-admit

🔄 Logic Flow

Overall Request Flow

```
┌──────────────────────────────────────────────────────────────┐
│                         HTTP Request                         │
└──────────────────────────────────────────────────────────────┘
                       │
                       ▼
┌──────────────────────────────────────────────────────────────┐
│                   FastAPI Route Handler                      │
│  • Parse request body                                        │
│  • Validate with Pydantic                                    │
│  • Extract parameters                                        │
└──────────────────────────────────────────────────────────────┘
                       │
                       ▼
┌──────────────────────────────────────────────────────────────┐
│                   Core Business Logic                        │
│  ┌──────────────────────────────────────────────────────┐    │
│  │  1. Data Extraction                                  │    │
│  │     • Parse command/file                             │    │
│  │     • Extract patient info                           │    │
│  └──────────────────────────────────────────────────────┘    │
│                      │                                       │
│  ┌───────────────────▼────────────────────────────────┐      │
│  │  2. HL7 Generation                                   │    │
│  │     • Try OpenAI GPT-4                          │   │
│  │     • Fallback to local generator               │   │
│  │     • Build HL7 segments                        │   │
│  └──────────────────┬──────────────────────────────┘   │
│                     │                                      │
│  ┌─────────────────▼──────────────────────────────┐    │
│  │ 3. Validation                                    │   │
│  │     • Parse HL7 structure                       │   │
│  │     • Check critical fields                     │   │
│  │     • Return validation result                  │   │
│  └──────────────────┬──────────────────────────────┘   │
│                     │                                      │
│  ┌─────────────────▼──────────────────────────────┐    │
│  │ 4. Mirth Integration (optional)                  │   │
│  │     • Wrap in MLLP envelope                     │   │
│  │     • Send via TCP socket                       │   │
│  │     • Receive ACK                               │   │
│  └──────────────────┬──────────────────────────────┘   │
└─────────────────────┬───────────────────────────────────┘
                      │
                      ▼
┌───────────────────────────────────────────────────────────┐
│                     JSON Response                            │
│  • HL7 message                                            │
│  • Validation results                                     │
│  • Status/errors                                          │
└───────────────────────────────────────────────────────────┘
```

HL7 Generation Logic

```
generate_hl7_message(client, command)
│
├─ Build prompt with current timestamp
│  "Generate HL7 message for: [command]
│   - Use HL7 v2.x format
│   - Auto-populate MSH-11='P', MSH-12='2.5'
│   - Include MSH, EVN, PID, PV1"
│
├─ Try OpenAI API
│  ├─ Call GPT-4o-mini
│  ├─ Temperature: 0.2 (deterministic)
│  ├─ Max tokens: 1500
│  ├─ Strip markdown formatting
│  └─ Return HL7 message
│
└─ Fallback (if OpenAI fails)
   ├─ Extract Patient ID: regex r"Patient ID[:\s]*([A-Za-z0-9\-_]+)"
   ├─ Extract Name: regex r"Patient Name[:\s]*([A-Za-z\-]+)\s+([A-Za-z\-]+)"
```

```
    ├─ Extract DOB: regex r"Date of
Birth[:\s]*([0-9]{4}[-/]?[0-9]{2}[-/]?[0-9]{2})"
    ├─ Extract Gender: regex r"Gender[:\s]*(M|F|Male|Female|U|Unknown)"
    ├─ Extract Address: regex r"Address[:\s]*([^\n]+)"
    ├─ Build MSH segment
    │
MSH|^~\&|SMART_APP|SMART_FAC|REC_APP|REC_FAC|[timestamp]||[trigger]||P|2.5
    ├─ Build EVN segment
    │  EVN|[event]|[timestamp]
    ├─ Build PID segment
    │  PID|1|[id]||[name]||[dob]|[gender]|||[address]
    ├─ Build PV1 segment
    │  PV1|1|[class]
    └─ Return joined segments
```

Validation Logic

```
validate_required_fields_api(hl7_message)
│
├─ Parse HL7 structure
│   ├─ Replace \n with \r (HL7 standard)
│   ├─ Call hl7.parse(message)
│   └─ Get list of segments
│
├─ For each segment (MSH, EVN, PID, PV1, etc.)
│   ├─ Check if segment is in CRITICAL_FIELDS
│   ├─ If yes, check required field indices:
│   │   ├─ MSH: field 9 (Message Type)
│   │   ├─ PID: fields 3, 5, 7, 8 (ID, Name, DOB, Sex)
│   │   ├─ ORC: fields 2, 3 (Order numbers)
│   │   ├─ OBR: fields 1, 2, 3, 4, 7 (Order info)
│   │   └─ OBX: fields 2, 3, 5 (Observation info)
│   └─ Add missing fields to list
│
└─ Return validation result
    ├─ is_valid: true/false
    ├─ missing_fields: [list]
    └─ message: "Validation passed" or "Missing critical fields"
```

📦 Data Models

HL7GenerationRequest

```
class HL7GenerationRequest(BaseModel):
    command: str              # Natural language command
    trigger_event: Optional[str] = "ADT-A01"  # HL7 trigger event
```

Example:

```
{
  "command": "Create patient John Doe",
  "trigger_event": "ADT-A01"
}
```

HL7ValidationResponse

```
class HL7ValidationResponse(BaseModel):
    is_valid: bool           # Validation passed or failed
    missing_fields: List[str] = []  # List of missing field IDs
    message: str             # Human-readable message
```

Example:

```
{
  "is_valid": false,
  "missing_fields": ["PID-3", "PID-7"],
  "message": "Missing critical fields"
}
```

HL7GenerationResponse

```
class HL7GenerationResponse(BaseModel):
    hl7_message: str         # Generated HL7 message
    validation: HL7ValidationResponse  # Validation results
    patient_info: Optional[Dict[str, Any]] = None  # Extracted info
```

MirthSendResponse

```
class MirthSendResponse(BaseModel):
    success: bool            # Send succeeded or failed
    message: str             # Status message
    acknowledgment: Optional[str] = None  # Mirth ACK message
```

BatchProcessingResponse

```
class BatchProcessingResponse(BaseModel):
    total_patients: int      # Total rows processed
    successful: int           # Successfully generated messages
    failed: int               # Failed generations
    messages: List[Dict[str, Any]]  # Individual results
```

🔧 Core Functions

1. `generate_hl7_message(client, command)`

Purpose: Generate HL7 message from text command

Parameters:

- `client`: ClientWrapper instance
- `command`: Natural language command string

Returns: HL7 message string

Logic:

1. Build prompt with formatting instructions
2. Try OpenAI API
3. Fallback to regex-based generator
4. Return HL7 message

Code Location: Lines 260-274

---

2. `validate_hl7_structure(hl7_message_text)`

Purpose: Validate HL7 message structure

Parameters:

- `hl7_message_text`: HL7 message string

Returns: Tuple (bool, parsed_message or error)

Logic:

1. Replace newlines with carriage returns

2. Parse with hl7.parse()
3. Return success/failure

Code Location: Lines 277-284

---

3. `validate_required_fields_api(hl7_message_text)`

Purpose: Validate critical fields (API version)

Parameters:

- `hl7_message_text`: HL7 message string

Returns: HL7ValidationResponse

Logic:

1. Parse HL7 structure
2. Check critical fields
3. Build validation response object

Code Location: Lines 286-318

---

4. `send_to_mirth(hl7_message, host, port)`

Purpose: Send HL7 to Mirth via MLLP

Parameters:

- `hl7_message`: HL7 message string
- `host`: Mirth hostname
- `port`: Mirth port (6661)

Returns: Tuple (success: bool, ack: str)

Logic:

1. Wrap message in MLLP envelope
2. Connect via TCP socket

3. Send message
4. Receive ACK
5. Parse ACK for AA/CA status

Code Location: Lines 321-350

---

5. `process_excel_batch(df, trigger_event)`

Purpose: Process entire Excel/CSV file

Parameters:

- `df`: Pandas DataFrame
- `trigger_event`: HL7 trigger event

Returns: List of result dictionaries

Logic:

1. Map columns flexibly
2. For each row:
    - Extract patient data
    - Generate HL7 message
    - Validate message
    - Add to results
3. Return all results

Code Location: Lines 353-436

⚠️ Error Handling

API Level Errors

```
try:
    # Process request
    result = generate_hl7_message(...)
except Exception as e:
    raise HTTPException(
        status_code=500,
        detail=f"Error generating HL7: {str(e)}"
    )
```

OpenAI Failures

```
if self.has_remote:
    try:
        return self.generate_via_api(...)
    except Exception as e:
        print(f"⚠ Remote generation failed: {e}")
        return fallback_hl7_generator(prompt)
```

Mirth Connection Errors

```
except socket.timeout:
    return False, "Connection timeout! Mirth may not be running"
except ConnectionRefusedError:
    return False, "Connection refused! Check Mirth channel"
except Exception as e:
    return False, f"Error: {str(e)}"
```

📝 Examples

Example 1: Generate Single HL7 Message

cURL:

```
curl -X POST "http://localhost:8000/api/generate-hl7" \
  -H "Content-Type: application/json" \
  -d '{
    "command": "Create ADT-A01 for John Doe, ID 12345, DOB 1985-03-15, Gender
M",
    "trigger_event": "ADT-A01"
  }'
```

Python:

```
import requests

response = requests.post(
    "http://localhost:8000/api/generate-hl7",
    json={
        "command": "Create ADT-A01 for John Doe, ID 12345, DOB 1985-03-15,
Gender M",
        "trigger_event": "ADT-A01"
    }
)
print(response.json())
```

Example 2: Upload Excel File

cURL:

```
curl -X POST "http://localhost:8000/api/upload-excel" \
  -F "file=@patients.xlsx" \
  -F "trigger_event=ADT-A01" \
  -F "send_to_mirth_flag=false"
```

Python:

```
import requests

with open('patients.xlsx', 'rb') as f:
    response = requests.post(
        "http://localhost:8000/api/upload-excel",
        files={'file': f},
        data={
            'trigger_event': 'ADT-A01',
            'send_to_mirth_flag': False
        }
    )
print(response.json())
```

Example 3: Validate HL7 Message

cURL:

```
curl -X POST "http://localhost:8000/api/validate-hl7" \
  -F
"hl7_message=MSH|^~\&|SMART_APP|SMART_FAC|REC_APP|REC_FAC|20251204|...\nEVN|A01
|...\nPID|1|12345|..."
```

Python:

```
import requests
```

```
hl7_msg =
"""MSH|^~\\&|SMART_APP|SMART_FAC|REC_APP|REC_FAC|20251204||ADT^A01||P|2.5
EVN|A01|20251204
PID|1|12345||Doe^John||19850315|M
PV1|1|I"""

response = requests.post(
    "http://localhost:8000/api/validate-hl7",
    data={'hl7_message': hl7_msg}
)
print(response.json())
```

Example 4: Send to Mirth

Python:

```
import requests

hl7_msg =
"""MSH|^~\\&|SMART_APP|SMART_FAC|REC_APP|REC_FAC|20251204||ADT^A01||P|2.5
EVN|A01|20251204
PID|1|12345||Doe^John||19850315|M
PV1|1|I"""

response = requests.post(
    "http://localhost:8000/api/send-to-mirth",
    data={'hl7_message': hl7_msg}
)
print(response.json())
```

🎓 Summary

Key Points:

1. Dual Mode: API and Console modes share 100% of business logic
2. Same Functions: All core functions are identical in both modes
3. AI-Powered: Uses OpenAI GPT-4 with automatic fallback
4. Flexible: Handles various Excel/CSV column formats
5. Validated: 3-tier validation system (system/contextual/critical)
6. Integrated: Direct Mirth Connect integration via MLLP

7. Well-Documented: Interactive Swagger UI at `/docs`

Architecture Benefits:

- ✅ Single source of truth for business logic
- ✅ Easy to maintain (one set of functions)
- ✅ Easy to test (test core functions once)
- ✅ Flexible interface (API or console)
- ✅ Well-documented (Swagger auto-generated)