# Technical Documentation

## AtlasBuilding_ChatBot

### Architecture Overview

The `AtlasBuilding_ChatBot` is a Python-based chatbot designed to answer natural-language questions with grounded evidence from CSV data using DuckDB and optionally a Retrieval-Augmented Generation (RAG) layer for explanations. The chatbot is built using the Streamlit framework and leverages various Python libraries and frameworks such as Django, Flask, FastAPI, SQLAlchemy, Pandas, NumPy, OpenAI, and LangChain.

### Setup & Installation

#### Prerequisites

- Python 3.8 or later
- Virtual environment (optional but recommended)

#### Steps to Set Up

1. **Clone the Repository**

```
git clone https://github.com/yourusername/AtlasBuilding_ChatBot.git
cd AtlasBuilding_ChatBot
```

2. **Create a Virtual Environment**

```
python -m venv venv
source venv/bin/activate  # On Windows use `venv\Scripts\activate`
```

3. **Install Dependencies**

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

4. **Set Up Environment Variables**

   - Create a `.env` file in the root directory with the following content:

```
# Model endpoint (Ollama via OpenAI API shim)
OPENAI_BASE_URL=http://localhost:11434/v1
OPENAI_API_KEY=ollama

# Chat model (Ollama tag)
OPEN_SOURCE_MODEL=qwen2.5:7b-instruct
```

```
# Embedding model (Ollama tag)
EMBED_MODEL=bge-m3

# Generation controls
LLM_TEMPERATURE=0.2

# Routing & Safety
FAST_INTENTS=1
FAST_SQL=1
LLM_ROUTING=1
STRICT_MODE=1  # use 1 for production; set 0 locally if you want small talk

MAX_ROWS=500

# Data & Storage
DATA_DIR=.
DUCKDB_PATH=atlas.duckdb
CHROMA_DIR=.chroma
CHROMA_COLLECTION=atlas-v2
```

5. **Run the Chatbot**

```
streamlit run src/ui.py
```

## API Documentation

The chatbot provides a RESTful API for querying data and interacting with the chatbot. The API endpoints are as follows:

- **GET /list_floors**: List all floors in the building.
- **GET /rooms_on_floor/{floor}**: List rooms on a specific floor.
- **GET /free_meeting_rooms_now/{floor}**: List free meeting rooms on a specific floor.
- **GET /utilization_by_floor/{days}**: Get utilization by floor for the last **`days`** days.
- **GET /busiest_rooms/{floor}/{days}/{limit}**: Get the busiest rooms on a specific floor for the last **`days`** days.
- **GET /peak_hours_by_floor/{floor}/{top}**: Get peak hours for a specific floor.

## Database Schema

The chatbot uses DuckDB to manage its data. The database schema includes the following tables:

- **events_office**: Stores office occupancy events.
- **events_meeting**: Stores meeting room occupancy events.
- **spaces**: Stores information about rooms and floors.
- **events_all**: Stores all occupancy events.

## Configuration

The chatbot's configuration is managed through environment variables. The `.env` file is used to set various configuration options such as model endpoints, API keys, and routing flags.

## Development Guidelines

- **Code Structure**: The code is organized into modules and packages within the `src` directory.
- **Testing**: Unit tests can be written using the `unittest` framework or `pytest`.
- **Documentation**: Documentation should be added to each module and function to explain its purpose and usage.
- **Version Control**: Use Git for version control and follow best practices for branching and merging.

## Deployment Instructions

1. **Build the Docker Image**

   ```
   docker build -t atlasbuilding_chatbot .
   ```

2. **Run the Docker Container**

   ```
   docker run -p 8501:8501 atlasbuilding_chatbot
   ```

3. **Access the Chatbot** Open a web browser and navigate to `http://localhost:8501` to access the chatbot.

## Conclusion

The `AtlasBuilding_ChatBot` is a powerful tool for answering natural-language questions with grounded evidence from CSV data. It leverages various Python libraries and frameworks to provide a robust and extensible solution for building management. With proper setup and configuration, the chatbot can be easily deployed and integrated into existing systems.