# Travel Information System

A comprehensive travel information system that combines flight and hotel data with a user-friendly web interface.

## System Architecture

This project consists of several integrated components:

* **Backend API**: A FastAPI application that handles requests for flight and hotel data
* **Frontend**: A Streamlit interface that provides user-friendly access to the data
* **Database Layer**:
  + **MongoDB**: Stores flight information in two collections (flights and flights\_segments)
  + **SQLite**: Stores hotel review information in two databases (hotel\_location.db and hotel\_rate.db)
* **Ollama**: Runs a local LLM (Large Language Model) for natural language processing

### Database Structure

#### MongoDB Collections

* **flights**: Contains flight information including departure/destination airports and pricing
* **flights\_segments**: Contains detailed flight segment information including airline names

#### SQLite Databases

* **hotel\_location.db**: Contains hotel location information (name, county, state)
* **hotel\_rate.db**: Contains hotel ratings and reviews (overall rating, sleep quality, service, etc.)

## Features

* **Flight Data**:
  + Search for flights by departure and destination airports
  + Search for flights by airline name
  + View all available flights
* **Hotel Data**:
  + Search for hotel reviews by county
  + Search for hotel reviews by state
  + Filter hotels by minimum rating
* **Frontend**:
  + Schema exploration for understanding database structure
  + Structured query interfaces for both flight and hotel data
  + Data modification capabilities (add, update, delete records)

## Prerequisites

* Docker and Docker Compose
* At least 8GB of RAM (for running the LLM)
* Approximately 10GB of disk space

## Setup Instructions

1. Clone this repository:

git clone https://github.com/yourusername/travel-information-system.git

cd travel-information-system

1. Make sure you have the SQLite databases in the data directory:

mkdir -p data

# Copy hotel\_location.db and hotel\_rate.db into the data directory

1. Start the application with Docker Compose:

docker-compose up -d

1. Wait for all services to start (this might take a few minutes on first run as the Ollama model is downloaded):

docker-compose logs -f

1. Access the Streamlit frontend at: <http://localhost:8501>
2. Pull Ollama manually

docker exec -it ollama /bin/bash

ollama pull llama3

## Using the Application

### Flight Search

You can search for flights by:

* Departure and destination airports
* Airline name
* Or view all available flights

### Hotel Search

You can search for hotels by:

* County
* State
* Minimum rating threshold

### Data Modification

The application also supports:

* Adding new flight and hotel records
* Updating existing records
* Deleting records

## Development

### Project Structure

project/

│

├── app/ # Backend application package

│ ├── \_\_init\_\_.py # Makes 'app' a Python package

│ ├── main.py # FastAPI backend entry point

│ ├── db\_config.py # Database configuration settings

│ ├── mongo\_agent.py # MongoDB interaction functions

│ └── sql\_agent.py # SQLite interaction functions

│

├── data/ # Directory for data files

│ ├── hotel\_location.db # SQLite database for hotel location

│ └── hotel\_rate.db # SQLite database for hotel ratings

│

├── docker-compose.yml # Docker Compose configuration

├── Dockerfile # Backend Docker configuration

├── Dockerfile.streamlit # Frontend Docker configuration

├── streamlit-requirements.txt # Frontend dependencies

├── streamlit\_app.py # Streamlit application

├── requirements.txt # Backend dependencies

└── README.md # Project documentation

### API Endpoints

#### Flight Endpoints

* GET /flights: Get all flights
* GET /flights/airports?starting={code}&destination={code}: Get flights between specific airports
* GET /flights/airline?airline={name}: Get flights operated by a specific airline

#### Hotel Endpoints

* GET /hotels: Get all hotel reviews (with optional filters)
* GET /hotels/county/{county}: Get hotel reviews for a specific county
* GET /hotels/state/{state}: Get hotel reviews for a specific state

### Environment Variables

The following environment variables can be customized:

* API\_URL: URL of the backend API (default: <http://backend:8000>)
* OLLAMA\_HOST: URL of the Ollama service (default: <http://ollama:11434>)
* MONGO\_URI: MongoDB connection string (default: MongoDB Atlas connection)
* MONGO\_HOST: MongoDB hostname (default: mongodb)
* MONGO\_PORT: MongoDB port (default: 27017)
* SQLITE\_DB\_DIR: Directory containing SQLite databases (default: ./data)

## Troubleshooting

### MongoDB Connection Issues

If you encounter issues connecting to MongoDB, check the following:

* Ensure your MongoDB connection string in db\_config.py is correct
* Verify the MongoDB container is running: docker-compose ps mongodb
* Check MongoDB logs: docker-compose logs mongodb

### Database File Issues

If the application can't find the SQLite databases:

* Make sure hotel\_location.db and hotel\_rate.db are in the data/ directory
* Verify file permissions allow the containers to read the files
* Try rebuilding the containers: docker-compose build --no-cache

### API Connection Errors

If the frontend can't connect to the API:

* Ensure all containers are running: docker-compose ps
* Check the API logs: docker-compose logs backend
* Verify that the API\_URL environment variable is set correctly in the frontend container