```
"""
Zoe - Real Estate Agent

"""

from typing import Optional, Dict, Any, List
from dataclasses import dataclass
from datetime import datetime
import os
import json
import requests
from loguru import logger
from swarms import Agent
from swarm_models import OpenAIChat
from dotenv import load_dotenv
from enum import Enum

# Configure loguru logger
logger.add(
    "logs/real_estate_agent_{time}.log",
    rotation="500 MB",
    retention="10 days",
    level="INFO",
    format="{time:YYYY-MM-DD at HH:mm:ss} | {level} | {message}",
)
```

```
class PropertyType(str, Enum):
    """Enum for property types"""

    OFFICE = "office"
    RETAIL = "retail"
    INDUSTRIAL = "industrial"
    MIXED_USE = "mixed-use"
    LAND = "land"


@dataclass
class PropertyListing:
    """Data class for commercial property listings"""

    property_id: str
    address: str
    city: str
    state: str
    zip_code: str
    price: float
    square_footage: float
    property_type: PropertyType
    zoning: str
    listing_date: datetime
    lat: float
```

```
    lng: float
    description: Optional[str] = None
    features: Optional[List[str]] = None
    images: Optional[List[str]] = None


class PropertyRadarAPI:
    """Client for PropertyRadar API integration"""

    def __init__(self, api_key: str):
        """Initialize PropertyRadar API client

        Args:
            api_key (str): PropertyRadar API key
        """
        self.api_key = api_key
        self.base_url = "https://api.propertyradar.com/v1"
        self.session = requests.Session()
        self.session.headers.update(
            {
                "Authorization": f"Bearer {api_key}",
                "Content-Type": "application/json",
            }
        )

    def search_properties(
```

```
    self,
    max_price: float = 10_000_000,
    property_types: List[PropertyType] = None,
    location: Dict[str, Any] = None,
    min_sqft: Optional[float] = None,
    max_sqft: Optional[float] = None,
    page: int = 1,
    limit: int = 20,
) -> List[PropertyListing]:
    """
    Search for commercial properties using PropertyRadar API

    Args:
        max_price (float): Maximum property price
        property_types (List[PropertyType]): Types of properties to search for
        location (Dict[str, Any]): Location criteria (city, county, or coordinates)
        min_sqft (Optional[float]): Minimum square footage
        max_sqft (Optional[float]): Maximum square footage
        page (int): Page number for pagination
        limit (int): Number of results per page

    Returns:
        List[PropertyListing]: List of matching properties
    """
    try:
        # Build the query parameters
```

```
params = {
    "price_max": max_price,
    "property_types": (
        [pt.value for pt in property_types]
        if property_types
        else None
    ),
    "page": page,
    "limit": limit,
    "for_sale": True,
    "state": "FL",  # Florida only
    "commercial_property": True,
}

# Add location parameters
if location:
    params.update(location)

# Add square footage filters
if min_sqft:
    params["square_feet_min"] = min_sqft
if max_sqft:
    params["square_feet_max"] = max_sqft

# Make the API request
response = self.session.get(
```

```
    f"{self.base_url}/properties",
    params={
        k: v for k, v in params.items() if v is not None
    },
)
response.raise_for_status()

# Parse the response
properties_data = response.json()

# Convert to PropertyListing objects
return [
    PropertyListing(
        property_id=prop["id"],
        address=prop["address"],
        city=prop["city"],
        state=prop["state"],
        zip_code=prop["zip_code"],
        price=float(prop["price"]),
        square_footage=float(prop["square_feet"]),
        property_type=PropertyType(prop["property_type"]),
        zoning=prop["zoning"],
        listing_date=datetime.fromisoformat(
            prop["list_date"]
        ),
        lat=float(prop["latitude"]),
```

```
                lng=float(prop["longitude"]),
                description=prop.get("description"),
                features=prop.get("features", []),
                images=prop.get("images", []),
            )
            for prop in properties_data["results"]
        ]

    except requests.RequestException as e:
        logger.error(f"Error fetching properties: {str(e)}")
        raise


class CommercialRealEstateAgent:
    """Agent for searching and analyzing commercial real estate properties"""

    def __init__(
        self,
        openai_api_key: str,
        propertyradar_api_key: str,
        model_name: str = "gpt-4",
        temperature: float = 0.1,
        saved_state_path: Optional[str] = None,
    ):
        """Initialize the real estate agent
```

```
    Args:
        openai_api_key (str): OpenAI API key
        propertyradar_api_key (str): PropertyRadar API key
        model_name (str): Name of the LLM model to use
        temperature (float): Temperature setting for the LLM
        saved_state_path (Optional[str]): Path to save agent state
    """
    self.property_api = PropertyRadarAPI(propertyradar_api_key)

    # Initialize OpenAI model
    self.model = OpenAIChat(
        openai_api_key=openai_api_key,
        model_name=model_name,
        temperature=temperature,
    )

    # Initialize the agent
    self.agent = Agent(
        agent_name="Commercial-Real-Estate-Agent",
        system_prompt=self._get_system_prompt(),
        llm=self.model,
        max_loops=1,
        autosave=True,
        dashboard=False,
        verbose=True,
        saved_state_path=saved_state_path,
```

```
            context_length=200000,
            streaming_on=False,
        )

        logger.info(
            "Commercial Real Estate Agent initialized successfully"
        )

    def _get_system_prompt(self) -> str:
        """Get the system prompt for the agent"""
            return """You are a specialized commercial real estate agent assistant focused on Central Florida properties.
        Your primary responsibilities are:
        1. Search for commercial properties under $10 million
        2. Focus on properties zoned for commercial use
        3. Provide detailed analysis of property features, location benefits, and potential ROI
        4. Consider local market conditions and growth potential
        5. Verify zoning compliance and restrictions

        When analyzing properties, consider:
        - Current market valuations
        - Local business development plans
        - Traffic patterns and accessibility
        - Nearby amenities and businesses
        - Future development potential"""
```

```
def search_properties(
    self,
    max_price: float = 10_000_000,
    property_types: List[PropertyType] = None,
    location: Dict[str, Any] = None,
    min_sqft: Optional[float] = None,
    max_sqft: Optional[float] = None,
) -> List[Dict[str, Any]]:
    """
    Search for properties and provide analysis

    Args:
        max_price (float): Maximum property price
        property_types (List[PropertyType]): Types of properties to search
        location (Dict[str, Any]): Location criteria
        min_sqft (Optional[float]): Minimum square footage
        max_sqft (Optional[float]): Maximum square footage

    Returns:
        List[Dict[str, Any]]: List of properties with analysis
    """
    try:
        # Search for properties
        properties = self.property_api.search_properties(
            max_price=max_price,
            property_types=property_types,
```

```
    location=location,
    min_sqft=min_sqft,
    max_sqft=max_sqft,
)

# Analyze each property
analyzed_properties = []
for prop in properties:
    analysis = self.agent.run(
        f"Analyze this commercial property:\n"
        f"Address: {prop.address}, {prop.city}, FL {prop.zip_code}\n"
        f"Price: ${prop.price:,.2f}\n"
        f"Square Footage: {prop.square_footage:,.0f}\n"
        f"Property Type: {prop.property_type.value}\n"
        f"Zoning: {prop.zoning}\n"
        f"Description: {prop.description or 'Not provided'}"
    )

    analyzed_properties.append(
        {"property": prop.__dict__, "analysis": analysis}
    )

logger.info(
    f"Successfully analyzed {len(analyzed_properties)} properties"
)
return analyzed_properties
```

```
        except Exception as e:
            logger.error(
                f"Error in property search and analysis: {str(e)}"
            )
            raise


def main():
    """Main function to demonstrate usage"""
    load_dotenv()

    # Initialize the agent
    agent = CommercialRealEstateAgent(
        openai_api_key=os.getenv("OPENAI_API_KEY"),
        propertyradar_api_key=os.getenv("PROPERTYRADAR_API_KEY"),
        saved_state_path="real_estate_agent_state.json",
    )

    # Example search
    results = agent.search_properties(
        max_price=5_000_000,
        property_types=[PropertyType.RETAIL, PropertyType.OFFICE],
        location={"city": "Orlando", "radius_miles": 25},
        min_sqft=2000,
    )
```

```
    # Save results
    with open("search_results.json", "w") as f:
        json.dump(results, f, default=str, indent=2)


if __name__ == "__main__":
    main()
```