# Solution

## Data Models

The following Pydantic models are used to represent trade records:

- **Trade**: Represents a trade record, including fields like **‘assetClass’**, **‘counterparty’**, **‘instrumentId’**, **‘instrumentName’**, **‘tradeDateTime’**, **‘tradeDetails’**, **‘tradeId’**, and **‘trader’**.
- **tradeDetails**: Represents the details of a trade, including the **‘buySellIndicator’**, **‘price’**, and **‘quantity’**.

## Endpoints

The Trade Management API provides the following endpoints:

- **GET ‘ /trades ’**: Retrieves a list of trade records based on specified query parameters.
  - Query Parameters:
    - **‘search’**: Search text across fields.
    - **‘assetClass’**: Asset class of the trade.
    - **‘start’**: Minimum date for tradeDateTime.
    - **‘end’**: Maximum date for tradeDateTime.
    - **‘minPrice’**: Minimum value for tradeDetails.price.
    - **‘maxPrice’**: Maximum value for tradeDetails.price.
    - **‘tradeType’**: Trade type (BUY or SELL).
    - **‘sort’**: Field to sort by.
    - **‘limit’**: Maximum number of trades to return (for pagination).
    - **‘offset’**: Number of trades to skip (for pagination).
  - Response: Returns a list of trade records that match the specified criteria.
- **GET ‘ /trades/{trade_id} ‘**: Retrieves a specific trade record by its trade ID.
  - Path Parameter: **‘trade_id’** the unique ID of the trade.
  - Response: Returns the trade record with the specified trade ID if found.

## Implementation Details

- The API utilizes FastAPI, a high-performance web framework, for handling HTTP requests and responses efficiently.
- Pydantic models are employed to provide data validation and enforce the structure of the input and output data.
- The trade records are stored in a dummy database, which is a list of dictionaries following a specific structure.
- The **'/trades'** endpoint applies the specified query parameters as filters, allowing users to search for trades based on various criteria.
- Sorting is implemented based on the **'sort'** parameter, enabling users to sort the trades by different fields.
- Pagination is achieved using the **'limit'** and **'offset'** parameters, allowing users to retrieve a subset of trades.
- The **'/trades/{trade_id}'** endpoint fetches a specific trade record by its trade ID from the database.

## Reasoning

- FastAPI was chosen as the web framework due to its high performance, modern features, and excellent support for async operations.
- Pydantic models were utilized to ensure data validation and enforce a consistent data structure throughout the API.
- The dummy database was used to simulate trade records and showcase the API's functionality without the need for a real database.
- The flexibility provided by query parameters allows users to customize their search and retrieve specific trade records efficiently.
- Sorting and pagination options enhance the usability of the API, enabling users to order and retrieve trade records in a controlled manner.
- The solution is designed to be scalable and adaptable, allowing for easy integration with a real database and additional features in the future.

By providing a robust and flexible Trade Management API, users can easily retrieve trade records based on their specific requirements, enabling efficient trade data management and analysis.