**Solution Description**

I have implemented a REST API using the FastAPI framework in Python to serve Trade data. The API provides endpoints for retrieving a list of Trades, retrieving a single Trade by ID, searching for Trades, and filtering Trades based on various criteria. The implementation includes support for pagination and sorting of Trades.

Here's a breakdown of the solution:

1. Mocked Database:
   * To simulate a data storage layer, I created a list called **trades\_db** to store the Trade objects.
   * When a new Trade is created using the **create\_trade** endpoint, it is appended to the **trades\_db** list.
2. Pydantic Models:
   * I defined the Pydantic models **Trade** and **TradeDetails** to represent the structure and validation rules for the Trade data.
   * These models ensure that the incoming requests adhere to the specified data schema.
3. API Endpoints:
   * **/trades** (GET): Retrieves a list of Trades.
     + The endpoint supports pagination through the **page** and **limit** query parameters, allowing the client to specify the page number and the number of Trades per page.
     + Sorting is implemented using the **sort** query parameter, which allows sorting by the tradeDateTime or tradeDetails.price fields in ascending or descending order.
     + Filtering is supported using optional query parameters such as **search**, **assetClass**, **end**, **maxPrice**, **minPrice**, **start**, and **tradeType**. Trades can be filtered based on the provided criteria.
     + The endpoint returns a paginated and filtered list of Trades based on the query parameters.
   * **/trades** (POST): Creates a new Trade.
     + The Trade object is received in the request body, validated against the Trade model, and added to the **trades\_db** list.
     + The created Trade is returned in the response.
   * **/trades/{trade\_id}** (GET): Retrieves a single Trade by its ID.
     + The Trade ID is provided as a path parameter.
     + The endpoint searches the **trades\_db** list for a Trade with a matching ID and returns it.
     + If no Trade is found, an HTTP 404 response is returned.
4. Implementation Approach:
   * I utilized the FastAPI framework due to its simplicity, high performance, and built-in support for validating request data using Pydantic models.
   * The use of Pydantic models ensures that the incoming request data is properly validated, reducing the risk of processing invalid or inconsistent data.
   * The implementation follows a modular approach, with separate functions for handling different endpoints and specific functionalities such as filtering, sorting, and pagination.
   * I used list comprehensions and lambda functions to filter and sort the Trades based on the specified criteria efficiently.
   * The solution allows flexibility in adding or modifying filtering and sorting options in the future by extending the query parameters and updating the corresponding logic.
5. Limitations and Future Improvements:
   * The solution utilizes a mocked database (**trades\_db**) for simplicity. In a real-world scenario, a proper data storage technology such as Elasticsearch would be used, and the implementation would need to be adapted accordingly.
   * Error handling and exception management can be further improved to provide meaningful error responses and handle edge cases.
   * Additional validation checks, such as ensuring unique Trade IDs, can be implemented to maintain data integrity.
   * Authentication and authorization mechanisms can be added to secure the API endpoints and restrict access based on user roles and permissions.
   * Automated unit tests and integration tests should be developed to ensure the correctness of the API and its functionalities.
   * Logging and monitoring capabilities should be added to track API usage and identify any issues or performance bottlenecks.

**This solution provides a functional REST API for Trade data retrieval, search, and filtering, meeting the requirements which was specified in the exercise. It is a starting point that can be further enhanced and adapted to meet real-world use cases and production requirements.**