I am providing the detailed implementation of the real-time alerting system prototype, including algorithm, data structures, and considerations for scalability, data integrity, and fault tolerance:

Algorithm and Data Structures:

1. Transaction Monitoring:
   - ➢ Maintain a data structure to store the transaction records in real-time. We can use a streaming platform such as Apache Kafka.
   - ➢ Process incoming transactions and perform real-time analysis to detect unusual activities.
2. Threshold Exceeding Alert:
   - ➢ For each incoming transaction, check if the transaction amount exceeds a pre-defined threshold.
   - ➢ If a transaction exceeds the threshold, trigger an alert for the corresponding syndicate and fund manager.
3. Spike in Transaction Rate Alert:
   - ➢ Maintain a sliding window data structure to track the transaction rate over a specified time interval (e.g., the last hour).
   - ➢ Calculate the average transaction rate within the sliding window.
   - ➢ For each incoming transaction, update the sliding window and check if the transaction rate exceeds a specified threshold (e.g., 10 times the average rate).
   - ➢ If a spike in the transaction rate is detected, trigger an alert for the corresponding syndicate and fund manager.

Pseudocode: The Pseudocode is available in the pseudocode.py of the task2 folder.

The pseudocode provided outlines the implementation of a real-time alerting system for monitoring transactions. It assumes the use of a streaming platform like Kafka to process incoming transaction data. Here's a breakdown of the pseudocode:

1. **Import Libraries:** The code begins by importing the necessary libraries, including the Kafka library and the datetime module.
2. **Threshold and Window Settings:** The pseudocode defines the threshold for the transaction amount exceeding alert (threshold), the sliding window size for the transaction rate (sliding_window_size), and the spike threshold for the transaction rate (spike_threshold).
3. **Configure Kafka Consumer:** The code configures the Kafka consumer to subscribe to the 'transactions' topic and connect to the Kafka server running on 'localhost:9092'.
4. **Sliding Window Initialization:** A sliding window data structure (sliding_window) is initialized to track the timestamps of transactions within the specified sliding window size.
5. **Monitor Transactions:** The monitor_transactions() function is defined as the main loop for processing incoming transactions.
6. **Process Transaction:** The process_transaction(transaction) function is called to process each incoming transaction.
7. **Check Threshold Exceeding:** The check_threshold_exceeding(transaction) function checks if the transaction amount exceeds the pre-defined threshold. If it does, the trigger_threshold_exceeding_alert() function is called to trigger an alert for the corresponding syndicate and fund manager.

8. **Check Transaction Rate Spike:** The check_transaction_rate_spike(transaction) function checks for a spike in the transaction rate. It updates the sliding window with the current timestamp, calculates the transaction rate using the calculate_transaction_rate() function, and triggers an alert using the trigger_transaction_rate_spike_alert() function if the transaction rate exceeds the average rate multiplied by the spike threshold.
9. **Update Sliding Window:** The update_sliding_window(current_timestamp) function adds the current timestamp to the sliding window and removes any timestamps that are outside the sliding window's time range.
10. **Calculate Transaction Rate:** The calculate_transaction_rate() function calculates the transaction rate within the sliding window by dividing the number of transactions in the window by the total time duration of the window.
11. **Trigger Threshold Exceeding Alert:** The trigger_threshold_exceeding_alert(syndicate, fund_manager) function is called to trigger an alert when a threshold-exceeding transaction is detected. It generates an alert message for the corresponding syndicate and fund manager.
12. **Trigger Transaction Rate Spike Alert:** The trigger_transaction_rate_spike_alert(syndicate, fund_manager) function is called to trigger an alert when a spike in the transaction rate is detected. It generates an alert message for the corresponding syndicate and fund manager.
13. **Send Alert:** The send_alert(message) function is a placeholder implementation to send the generated alert message to the intended recipients. You'll need to replace this function with your actual alerting mechanism.
14. **Parse Transaction:** The parse_transaction(message) function is a placeholder that you need to implement. It should parse the incoming transaction message and extract the relevant fields such as amount, syndicate, and fund manager.
15. **Start Monitoring Transactions:** The consumer starts monitoring transactions by calling the monitor_transactions() function.

The provided pseudocode serves as a guideline for implementing a real-time alerting system.

Scalability, Data Integrity, and Fault Tolerance:

Scalability:

- To handle scalability, distribute the transaction processing across multiple instances or nodes.
- Utilize stream processing frameworks that provide scalability and fault tolerance out of the box, like Apache Kafka or Apache Flink.

Data Integrity:

- Ensure that the transaction data is stored securely and cannot be tampered with.
- Implement appropriate authentication and authorization mechanisms to protect the data.

Fault Tolerance:

- Use replication and data backup strategies to ensure data availability in case of failures.
- Implement fault-tolerant stream processing frameworks that can handle failures and ensure data consistency.