Algorithm and Data Structures:

1. **Data Ingestion:**
   - ➔ Incoming transaction data is streamed into the system in real-time.
   - ➔ Data can be organized into a distributed message queue, such as Apache Kafka, for efficient ingestion.

2. **Data Storage:**
   - ➔ Maintain a distributed, high-throughput, and low-latency database for storing transaction data.
   - ➔ Use columnar storage databases like Apache Cassandra or time-series databases like InfluxDB.
   - ➔ Ensure data partitioning for horizontal scalability.

3. **Threshold Alerts:**
   - ➔ Maintain a separate data structure to store pre-defined transaction thresholds for each syndicate.
   - ➔ When a new transaction is received, compare its amount to the threshold for the respective syndicate.
   - ➔ Trigger an alert if the transaction amount exceeds the threshold.
   - ➔ Use in-memory data structures (e.g., hash maps) for quick threshold lookups.

4. **Transaction Rate Alerts:**
   - ➔ Calculate the moving average of transaction rates for each syndicate over a specific time window (e.g., one hour).
   - ➔ Compare the incoming transaction rate to this moving average.
   - ➔ Trigger an alert if the transaction rate spikes significantly (e.g., 10x the average).
   - ➔ Implement this with sliding windows and a data structure like a priority queue.

Scalability, Data Integrity, and Fault Tolerance

1. **Scalability:**
   - ➔ Use a distributed data processing framework like Apache Flink or Apache Kafka Streams to handle the high data velocity.
   - ➔ Horizontal scaling of the system to accommodate increased syndicates and transaction data.

2. **Data Integrity:**
   - ➔ Implement data validation and cleansing to ensure the correctness of incoming data.
   - ➔ Use distributed databases with built-in replication and consistency mechanisms to maintain data integrity.

3. **Fault Tolerance:**
   - ➔ Deploy the system across multiple data centers or cloud regions to mitigate single points of failure.
   - ➔ Implement failover mechanisms to handle hardware or software failures.
   - ➔ Use backup and recovery processes for data storage to prevent data loss.

4. **Monitoring and Logging:**
   - ➔ Implement extensive logging and monitoring to detect and respond to issues in real-time.
5. **Security:**
   - ➔ Ensure secure data transmission and access controls to protect sensitive transaction data.

In summary, this prototype outlines the algorithm and data structures for a real-time alerting system for syndicate transaction monitoring. It focuses on efficiency, scalability, reliability, and real-time processing challenges. The system employs distributed databases, data streaming, and alerting mechanisms to handle various scenarios while maintaining data integrity and fault tolerance.