Problem 2

Algorithm and Data Structures:

For single transaction alerting:

- For tracking a single transaction amount exceeding a pre-defined threshold, you can use a simple key-value store (e.g., Redis) where the key is the syndicate or user ID, and the value is the total transaction amount.
- For each incoming transaction, update the transaction amount for the corresponding syndicate.
- Send an alert (e.g, Email, SMS) if the transaction amount exceeds the pre-defined threshold.

Average rate-based alerting for sudden spike:

- To detect sudden spikes in the number of transactions, we can use a sliding window algorithm.
- Maintain a sliding window of transactions, e.g., using a circular buffer or a time-based database.
- Calculate the average transaction rate over the last hour.
- If the incoming transaction rate is significantly higher (e.g., 10x) than the average rate, send an alert (e.g, Email, SMS).

Code:

Here's a simplified PHP code to give you an idea of how the system might work:

```php
$threshold = 10000; // Pre-defined threshold for single transaction amount
$windowSize = 3600; // Size of the sliding time window (1 hour)
$transactions = array(); // Array to store transactions within the window

// Function to check for threshold-based alerts
function checkThreshold($transaction, $threshold) {
    if ($transaction['amount'] > $threshold) {
        // Send alert to fund manager
        return "Alert: Single transaction exceeds threshold - " . $transaction['amount'];
    }
    return null;
}

// Function to check for rate-based alerts
function checkRate($transactions, $windowSize) {
    $current_time = time();
    $window_start_time = $current_time - $windowSize;
    $window_transactions = array_filter($transactions, function($transaction) use
($window_start_time) {
        return $transaction['timestamp'] >= $window_start_time;
    });

    if (count($window_transactions) >= 10 * (count($transactions) / $windowSize)) {
        // Send alert to fund manager
        return "Alert: Sudden spike in transaction rate";
    }
    return null;
}

// Main loop to continuously monitor transactions
while (true) {
    $transaction = get_next_transaction(); // Get stored transaction into redis in real-time

    // Add the transaction to the list
    $transactions[] = $transaction;

    // Check for threshold-based alerts
    $threshold_alert = checkThreshold($transaction, $threshold);
    if ($threshold_alert) {
        // Send alert to fund manager
        send_alert($threshold_alert);
    }

    // Check for rate-based alerts
```

```
    $rate_alert = checkRate($transactions, $windowSize);
    if ($rate_alert) {
        // Send alert to fund manager
        send_alert($rate_alert);
    }

    // Remove old transactions from the list
    $current_time = time();
    $transactions = array_filter($transactions, function($transaction) use ($current_time,
$windowSize) {
        return $transaction['timestamp'] >= $current_time - $windowSize;
    });
}
```

Scalability:

- Scale database properly (like, indexding, partitioning and sharding)
- Use a distributed stream processing technology like, RabbitMQ or Apache Kafka or other tools to handle high volumes of real-time data.
- User load balancing and partitioning to distribute the workload across multiple processing nodes (In server-side)
- Decompose the system into smaller, independent microservices that can handle requests individually.

Data Integrity:

- To validate incoming transaction before processing payment
- Use reliable data storage (e.g, Redis) to store the transaction history and syndicate thresholds.

Fault Tolerance:

- Implement redundancy and replication in the processing nodes to handle node failures gracefully. (For example, k8s)

- Implement checkpoints and reliable storage methods to restore the system's condition if problems occur.

- Set Up a monitoring system which can check the system continuously and notify authority.