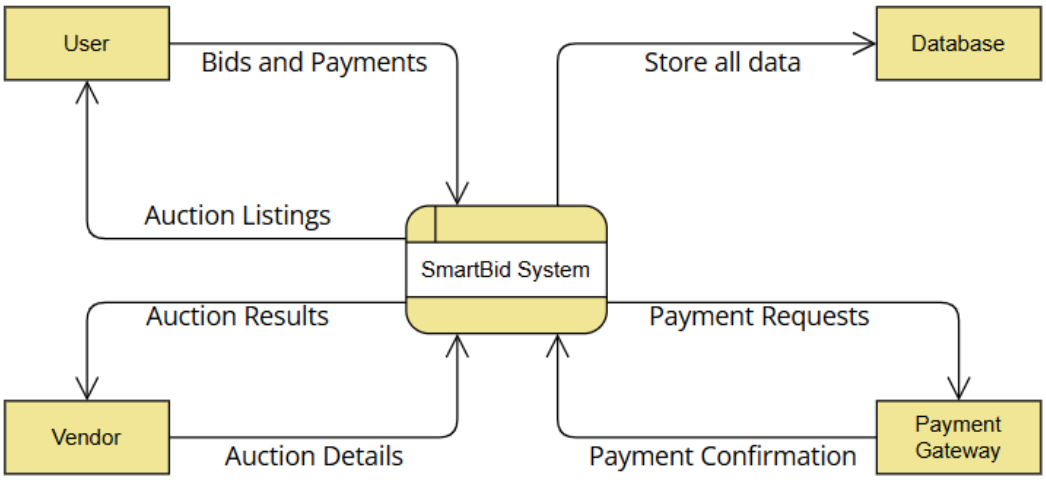
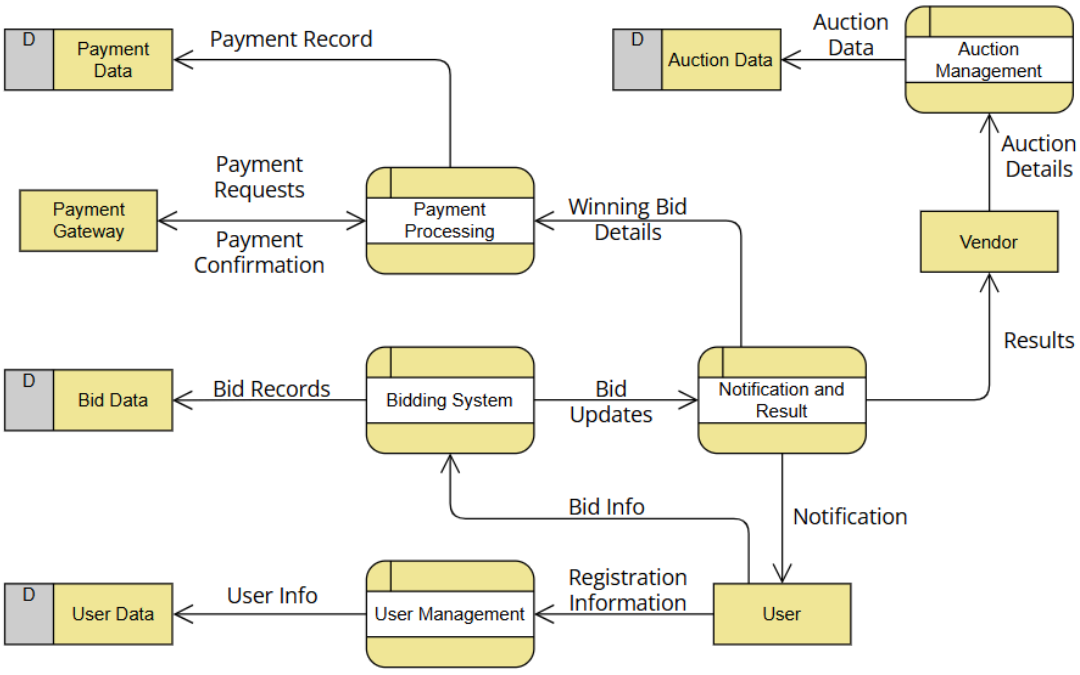
29 Rishi Kadam
40 Aayush Mohite
41 Manas More

# SLA - DFD & RMMM

## Level 0 DFD Diagram :




## Level 1 DFD Diagram :



## RMMM Sheet :

**Risk1: Software Integration Risk**

| **Risk Information Sheet** | | | |
|---|---|---|---|
| **Risk ID :** 1 | **Date :** 15/10/2025 | **Prob :** 70% | **Impact :** High |

**Description :**

1. Concurrency Challenges: During high-traffic auctions, a large number of simultaneous bids can create race conditions, where the system struggles to process overlapping bid updates correctly.
2. Data Integrity: Failure to manage concurrent bid transactions can corrupt auction data, leading to incorrect winner determination, inaccurate bid history, and a loss of user trust.
3. Performance Bottlenecks: The intense load during the final moments of a popular auction can overwhelm the server and database, increasing the chance of errors and system failures

**Refinement / Context :**

- **Sub Condition 1:** Two users submit bids at nearly the same instant. The system processes the lower bid last due to a race condition, incorrectly overwriting the actual highest bid.
- **Sub Condition 2:** A database transaction fails because of a deadlock between two competing bids, but the error is not handled correctly, leaving the auction in an inconsistent state.
- **Sub Condition 3:** The user interface confirms a successful bid to a user, but the bid was never committed to the database due to a server error, causing the user to mistakenly believe they are the highest bidder.

**Mitigation / monitoring :**

1. Use transactional database operations (ACID properties) to ensure that each bid is processed as a single, atomic unit.
2. Implement robust server-side validation and locking mechanisms to serialize bid submissions on critical auction items.

3. Conduct regular code reviews focused on the bidding logic and concurrency handling.
4. Perform rigorous stress testing with simulated high-concurrency scenarios to identify and fix performance bottlenecks before deployment.
5. Continuously monitor application and database logs for transaction failures, deadlocks, and other concurrency-related errors.

**Management / contingency plan/ trigger:**

Trigger: Detection of data inconsistencies via monitoring tools or a user dispute.

Plan: Establish a clear, documented dispute resolution process managed by an administrator who can manually verify bid logs. All bid attempts and system actions must be logged for auditability. If significant data corruption occurs, the plan includes immediately pausing the affected auction, notifying participants, and restoring the auction's state from the last known valid point using database logs and backups.

**Current status :** Monitoring in progress during development

| | |
|---|---|
| **Originator :** Manas, Ayush, Rishi | **Assigned :** Manas, Ayush, Rishi |

**Risk2: Third-Party Payment Gateway Integration Risk**

| **Risk Information Sheet** | | | |
|---|---|---|---|
| **Risk ID :** 2 | **Date :** 15/10/2025 | **Prob :** 60% | **Impact :** High |

**Description :**

The successful completion of an auction is entirely dependent on the functionality and reliability of the third-party payment gateway. Any issue with the integration or the gateway itself poses a direct risk to the core business process of the application (payment processing for winning bidders )

**Refinement / Context :**

- Sub Condition 1 (Technical): The payment gateway's API changes without adequate notice, or the documentation provided is inaccurate or incomplete, leading to delays and errors in implementation and payment processing.
- Sub Condition 2 (Operational): The payment gateway experiences intermittent downtime or slow response times, causing successful payment confirmation to the user interface to take longer than the performance requirement of 2 seconds for bid updates (which can be extended to payment status). This leads to a poor user experience and potential payment failures.
- Sub Condition 3 (Security/Compliance): The chosen payment gateway is not compliant with the latest security standards, or the integration method exposes SmartBid to security vulnerabilities, contradicting the requirement for securely handled payment information.

**Mitigation / monitoring :**

1. **Mitigation:** Select a widely-used, highly-rated payment gateway with stable APIs, excellent documentation, and robust security certifications (e.g., PCI DSS compliance). Implement comprehensive error handling and fallback mechanisms for API failures (e.g., retries).
2. **Monitoring:** Continuously monitor transaction logs for payment processing time, success/failure rates, and specific error codes returned by the gateway. Set up automated alerts for high failure rates or excessive latency in payment confirmation.

**Management / contingency plan/ trigger:**

Trigger: Consistent increase in payment transaction failures (e.g., 5% increase over a 24-hour period) or a sudden, unexplained lag in payment confirmation times.

Plan: Immediately implement a backup payment method or a manual payment verification process involving the administrator. Communicate clearly with winning bidders about the temporary issue and expected resolution time. Begin immediate investigation and engagement with the third-party payment gateway's technical support team.

**Current status :** Monitoring in progress during development

| | |
|---|---|
| **Originator :** Manas, Ayush, Rishi | **Assigned :** Manas, Ayush, Rishi |

**Risk3: Real-Time Communication Performance Degradation**

| Risk Information Sheet | | | |
|---|---|---|---|
| **Risk ID :** 3 | **Date :** 15/10/2025 | **Prob :** 75% | **Impact :** High |

**Description :**

The system requires real-time updates for bid changes, and the SRS specifies a performance requirement that bid updates must reflect within **2 seconds**. The reliance on WebSocket or polling for real-time changes combined with a concurrent user load of 100+ users creates a high risk that the communication infrastructure will become a bottleneck, leading to degraded performance.

**Refinement / Context :**

1. Sub Condition 1 (Server Load): The chosen real-time communication method (especially polling for many concurrent users) places a massive, unmanageable load on the Node.js backend and the MySQL database , preventing the system from handling 100 concurrent users without performance degradation.
2. Sub Condition 2 (Latency): Due to inefficient handling of the real-time feed, a bid is placed, but the update takes longer than the required 2 seconds to appear on other users' screens, leading to users placing bids based on outdated information.
3. Sub Condition 3 (Client-Side Performance): The constant stream of real-time data overwhelms the browser on the client-side, causing the user interface (developed with React ) to become slow, unresponsive, or laggy, hindering a user's ability to place bids promptly.

**Mitigation / monitoring :**

1. **Mitigation:** Use an efficient, event-driven architecture (like WebSockets ) for broadcasting bid changes, rather than less efficient polling. Implement server-side logic to only broadcast relevant updates (e.g., to users viewing the specific auction) to minimize data traffic. Optimize the database query for fetching the current top bids.
2. **Monitoring:** Monitor the application's network latency between the backend and the browser. Track the end-to-end time from a bid submission to the broadcast of the bid

update to other clients to ensure the 2-second limit is met. Monitor server CPU and memory usage during stress testing.

**Management / contingency plan/ trigger:**

**Trigger:** Consistent failure to meet the **2-second bid update reflection time** during internal stress testing or a drop below the required 100 concurrent user handling capacity.

**Plan:** If the performance requirement is unmet, temporarily reduce the frequency of real-time updates (e.g., increase polling interval or slow WebSocket pushes) and display a warning banner to users about the high-load conditions. The long-term plan is to migrate the real-time logic to a dedicated, high-performance service (like Redis or a specialized real-time database) to offload the strain from the main Node.js/MySQL stack.

**Current status :** The communication method is still TBD (WebSocket/polling)

| | |
|---|---|
| **Originator :** Manas, Ayush, Rishi | **Assigned :** Manas, Ayush, Rishi |