System Model Description

Overall Architecture and Request Flow

- **Frontend to Orchestrator:**
  The web frontend displays a checkout page. When a user submits an order, a POST request with order details in JSON format is sent to the Orchestrator. The Orchestrator serves as the entry point, coordinating validation and processing by communicating with backend services using gRPC.
- **Orchestrator to Backend Services:**
  Upon receiving an order, the Orchestrator sequentially or concurrently triggers a series of partial checks by sending gRPC requests to three core services:
    - **Transaction Verification Service:** Validates the order by checking that items are present, required user fields are filled, and the credit card format is correct.
    - **Fraud Detection Service:** Performs both basic and advanced fraud checks—using simple logic (e.g., name matching) and an XGBoost model—to detect suspicious activity.
    - **Suggestions Service:** Analyzes the order content to generate recommended book suggestions.

If all validations pass, the Orchestrator enqueues the order into the Order Queue Service.

- **Order Queue Service and Executors:**
  The Order Queue Service maintains an in-memory priority queue where orders are stored based on criteria (such as shipping country). The Executor services, which run in replicated mode for fault tolerance, are responsible for the final order processing. They use a leader election algorithm to elect a leader—only the leader dequeues orders from the queue, ensuring that only one order is processed at a time.

Roles of Each Microservice

- **Transaction Verification Service:**
  Validates core order details by ensuring the presence of items, the completeness of user information, and proper credit card formatting (e.g., via Luhn's algorithm).

- **Fraud Detection Service:**
  Analyzes the order for potential fraud using basic checks (e.g., comparing user names) and advanced techniques (e.g., an XGBoost model).
- **Suggestions Service:**
  Provides book recommendations based on order content, transforming order details into a list of suggested titles.
- **Order Queue Service:**
  Acts as a staging area by storing approved orders in an in-memory priority queue, prioritizing orders (for example, by shipping country).
- **Executor Service:**
  Processes orders for final execution. Multiple executor instances use a Bully election algorithm and heartbeat monitoring to ensure that only one instance (the leader) dequeues orders from the queue. The leader sends periodic heartbeats, and if a follower detects a missing heartbeat, it triggers an election to establish a new leader.

Vector Clocks for Event Ordering

Vector clocks are implemented across key validation events to maintain the causal order in our distributed system. Each service (Transaction, Fraud, Suggestions) updates its assigned slot in the vector clock upon processing an order, and the Orchestrator merges these clocks to create a consistent global view. This mechanism helps trace the sequence of events and diagnose any issues related to ordering or concurrency.

Leader Election and Mutual Exclusion

The Executor service uses a simplified Bully algorithm for leader election:

- **Unique IDs:** Each executor is assigned a unique ID (via an environment variable), and all peer IDs are known.
- **Heartbeat Mechanism:** The current leader sends heartbeats periodically; followers monitor these. If a heartbeat is missed beyond a timeout, an election is triggered.
- **Election Process (Bully Algorithm):**
  When a follower detects a missed heartbeat, it sends election messages to peers

with higher IDs. If no higher peer responds, it becomes the leader; otherwise, the highest-ID node is elected as the new leader.

- **Mutual Exclusion:**
  Only the leader dequeues orders from the Order Queue, ensuring that no two executors process the same order concurrently.

Logging and Error Handling

- **Logging:**
  Each microservice logs critical events. The Orchestrator logs incoming requests, outgoing gRPC calls (with vector clocks and order IDs), and the responses or errors returned by each service. The Executor logs heartbeat messages, election events, and order processing steps. These logs help trace the detailed flow of operations and facilitate troubleshooting.

- **Error Handling:**
  All gRPC calls are wrapped in try/except blocks. When an error occurs (such as a failure during a validation check or an RPC call), the error and its traceback are logged in detail. The orchestrator then returns a structured JSON error response to the frontend, enabling consistent and traceable error propagation throughout the system.