# Simulator Requirements

1. **User Interface** for the simulator. Can be built using react, vue.js, or whatever Ishan is using.

* Functionality: Display real-time data, accept user inputs for buy/sell orders, show portfolio, and provide simulation controls.

1. **Backend Server:**
   1. **Programming Language:** Python, due to its extensive libraries for financial analysis and ease of use.
   2. **Framework:** Flask or Django for handling HTTP requests and managing the application logic.
2. **Data Management:**
   1. **Database:** SQL (like PostgreSQL or MySQL) for structured data storage, like user accounts and transaction history.
   2. **In-Memory Data Store:** Redis or similar for caching and quick access to frequently used data like current market prices.
3. **Market Data Feed:**
   1. **Data Source:** Real-time or historical market data API (like Alpha Vantage, Yahoo Finance, or a custom data provider).
   2. **Data Processing:** Pandas for data manipulation and analysis.
4. **Trading Engine:**
   1. **Order Processing:** Simulate order execution, including order matching, execution algorithms, and handling different order types.
   2. **Risk Management:** Implement risk checks and constraints, like stop-loss or maximum exposure limits.
5. **Analytics and Reporting:**
   1. **Analytics:** Use Python libraries like NumPy, SciPy for statistical analysis, and Matplotlib or Plotly for data visualization.
   2. **Reporting:** Generate reports on trading performance, P&L statements, and other relevant metrics.
6. **Integration and Middleware:**
   1. **Message Queues:** RabbitMQ or Kafka for asynchronous task processing and inter-service communication.
   2. **APIs:** RESTful APIs for interfacing between different components.
7. **Testing and Simulation Control:**
   1. **Backtesting:** Implement backtesting capabilities using historical data to validate trading strategies.
   2. **Control Layer:** Logic to start, stop, and configure the simulation parameters.
8. **DevOps and Infrastructure:**
   1. **Containerization:** Docker for containerizing the application, ensuring consistency across different environments.
   2. **Orchestration:** Kubernetes or Docker Compose for managing the deployment of the containers.
   3. **CI/CD:** Continuous Integration and Continuous Deployment using tools like Jenkins or GitLab CI/CD.
9. **Security and Compliance:**
   1. **Authentication and Authorization:** Implement secure login and access control.
   2. **Data Encryption:** Ensure data is encrypted in transit and at rest.
   3. **Compliance:** Adhere to relevant financial regulations and best practices.

**-------------------------------------------------------------------------------------------------------------------------------**

To understand how each component is connected in the trading engine architecture, let's walk through the flow of information and processes:

1. **User Interface (UI) - Order Placement:**
   * **Connection to Backend:** The UI captures the user's order details and sends them to the backend server via an API call.
2. **Order Validation:**
   * **Receiving Orders:** The backend server receives the order from the UI and initiates validation processes.
   * **Communication with Database:** It queries the database to validate user authentication and checks if the order complies with trading rules.
3. **Risk Management Check:**
   * **Post-Validation:** After an order passes validation, it goes through risk management checks.
   * **Database Queries:** The server checks against the user's current positions and account details in the database to ensure the order does not breach any risk thresholds.
4. **Order Queue and Database Entry:**
   * **Order Queueing:** If the order passes risk management checks, it's placed in a processing queue.
   * **Database Recording:** Concurrently, the order details are recorded in the database, marking the order as pending.
5. **Order Matching and Execution Algorithms:**
   * **Dequeue and Processing:** Orders are dequeued according to the trading algorithm's logic.
   * **Matching Algorithm:** The system matches orders based on the current market data feed, which may be simulated or real-time data.
   * **Execution Algorithm:** Determines the specifics of how orders are executed based on predefined logic or historical data patterns.
6. **Order Execution:**
   * **Database Update:** Once an order is executed, the database is updated to reflect this, changing the order's status.
   * **User Portfolio Update:** The changes are also applied to the user's portfolio information stored in the database.
7. **Post-Execution Processing:**
   * **Notifications:** The system sends a trade confirmation to the UI, which can be displayed to the user.
   * **Portfolio and Balance Update:** The user's portfolio and account balance are updated in the database.
8. **UI Update:**
   * **Reflect Changes:** The UI polls or receives updates from the backend to show the latest status of the order and any portfolio changes.
   * **Market Data Display:** The UI also updates to show real-time or simulated market data relevant to the user's interests.
9. **Reporting and Analytics:**
   * **Data Aggregation:** The backend aggregates data from the database to prepare reports and analytics.
   * **Analytics Delivery:** The processed information is sent to the UI for the user to review.
10. **Continuous Monitoring and Risk Management:**
    * **Ongoing Checks:** The system continuously monitors all open orders and positions for risk management purposes.
    * **Alerts and Adjustments:** It applies adjustments to positions or sends alerts to users based on predefined criteria or detected risk conditions.

Here's how you could visually represent these connections in a diagram:

* The UI is the starting point where the user interacts.
* Arrows from the UI lead to the backend server, indicating the flow of order data.
* The server connects to the database, which is a central repository for orders, users, and transactions.
* Risk management checks loop between the server and the database to validate each order.
* The order processing queue is shown as a stage between risk checks and order execution.
* The execution of orders leads to updates in the database, which then flow back to the user's portfolio on the UI.
* Continuous monitoring is a feedback loop from the database to the trading engine, indicating ongoing risk checks.

A diagram of a process flow

Description automatically generated