Optimizing System Performance

Performance metrics:

- Response Time
- Latency
- Scalability
- Availability

Caching API Responses:
For ChatGPT API integration, a primary metric is the API response time, which averages between 15-20 seconds. Minimizing this delay through caching and system optimizations is a top priority to ensure a seamless user experience. Frequently requested responses from ChatGPT can be cached using in-memory storage solutions like Redis. By storing responses for repeated queries, the system can serve cached data instantly, avoiding repeated API calls and saving the 15-20 seconds per request. Similarly, other frequently accessed data can also be cached, such as chatbot configurations, or course and assignment details.

Rate Limiting:
To manage API usage effectively and prevent overloading the backend, rate limiting can be implemented. By controlling the number of requests made per user within a specified timeframe, rate limiting ensures that API calls remain within allocated quotas and that the service operates reliably under large traffic levels.

Database Sharding:
Database sharding can be utilized to improve scalability and reduce latency by partitioning data across multiple servers. This approach ensures that large datasets, such as user chatbot logs, are distributed evenly, reducing query load on individual database instances. This may not be necessary depending on the number of users.

Complete a Tutorial
The tutorial that I selected was focused around React hooks. I learned how custom hooks can encapsulate reusable logic, making components cleaner and more maintainable, which will help with fetching data from the backend in a modular way. The tutorial also covered hooks such as the useMemo hook, highlighting its role in optimizing performance by memoizing computationally expensive functions, preventing unnecessary re-renders. This will help to improve the web interfaces efficiency and code maintainability.