Advanced Functionality Report

Project: MicroCourses Full-Stack Web Application

1. Caching Strategy

To ensure the application remains performant as the number of users and courses grows, a caching strategy would be implemented for high-traffic endpoints such as the course listing.

In development, this can be done using in-memory caching. In production, a more robust solution like Redis would be adopted. Redis allows data to be temporarily stored in memory, which significantly reduces the load on the database and improves response times for frequent requests.

Cache invalidation would occur whenever a new course is added or deleted. Additionally, a time-to-live (TTL) mechanism would ensure that stale data is automatically purged after a set period, even if invalidation fails. This balance of manual invalidation and automatic expiry maintains both performance and data accuracy.

2. Load Balancing

To support multiple concurrent users and ensure scalability, load balancing would be used to distribute incoming HTTP requests evenly across multiple instances of the Express server.

In production, an HTTP reverse proxy such as Nginx or a cloud-based load balancer (e.g., AWS Elastic Load Balancer) would be configured. This would allow requests to be routed to the least busy or most responsive server instance, reducing the risk of overload on a single node.

This setup not only improves performance under load but also enhances fault tolerance. If one instance fails, traffic can automatically be rerouted to the remaining healthy instances, ensuring continuous availability.

3. High Availability Strategy

Backend (Express Application)
The backend can achieve high availability by running multiple instances of the Express server using a process manager like PM2 or containerization via Docker. These instances can be deployed across different servers or availability zones in a cloud environment. A load balancer ensures even traffic distribution and seamless failover.

To handle unexpected crashes or system errors, process monitoring tools would automatically restart failed instances. Cloud hosting platforms like Heroku, Render, or AWS EC2 also offer features such as auto-scaling and health checks to further improve resilience.

Database (MongoDB Atlas)
MongoDB Atlas natively supports high availability through replica sets. A typical configuration includes a primary node and two secondary nodes. If the primary node becomes unavailable, one of the secondaries is automatically promoted to primary with no manual intervention.

Atlas also handles automated backups and supports point-in-time recovery, ensuring data safety even during critical failures. This setup guarantees that the course data remains available and consistent at all times.

Summary

The MicroCourses application has been designed with performance and resilience in mind. Through caching, load balancing, and high availability strategies, it is capable of scaling effectively while maintaining reliability and responsiveness.

These strategies align with best practices for full-stack web applications and ensure that the application can support both current and future demands as it evolves.