**SYSTEM DESIGN**

To create a real-time, in-game basketball statistics application for the coach, I would design a scalable and reliable system that can handle up to 100 users simultaneously. Here's an overview of the system architecture:

**System Architecture:**

**1. Frontend Application:**

- The frontend will be developed using modern web technologies such as Angular for the user interface.

- The application will allow users to navigate between ongoing games and historical games seamlessly.

* **Web Framework:** The frontend will be developed using a modern web framework Angular. These frameworks provide a rich user interface and are well-suited for building responsive web applications.
* **User Interface (UI) Design:** The UI should be designed to be user-friendly and intuitive, allowing users to easily navigate between ongoing games and historical data. It should also provide real-time updates during games.
* **Mobile Responsiveness:** Ensure that the application is mobile-responsive to cater to users accessing the app from different devices.

**2. Backend Services:**

- Backend services will be hosted on a cloud infrastructure like AWS, Azure, or Google Cloud for scalability and reliability.

- Node.js or Python can be used to build the backend server due to their efficiency and event-driven capabilities.

- A load balancer will distribute incoming requests among multiple backend instances to ensure high availability.

* **Server**: Use Node.js or Python for the backend server due to their event-driven nature, which is well-suited for handling real-time updates.
* **Load Balancer**: Implement a load balancer (e.g., AWS Elastic Load Balancing) to distribute incoming requests among multiple backend server instances. This ensures high availability and prevents any single point of failure.
* **Application Logic**: The backend should handle user authentication, game data retrieval, and real-time data streaming to clients. Game data can be obtained through APIs or from a database.

**3. Real-time Data Streaming:**

- WebSocket technology will be used to provide real-time updates to users during games. This ensures low latency and instant data delivery.

- A message broker like Apache Kafka or RabbitMQ can be used to handle the communication between the server and clients.

* **WebSocket**: Implement WebSocket technology for real-time updates. WebSocket connections are bi-directional and provide low-latency communication between the server and clients.
* **Message Broker**: Use a message broker like Apache Kafka or RabbitMQ to manage communication between the backend server and WebSocket connections. This decouples the sender and receiver and ensures reliable message delivery.

**4. Database:**

- For historical game data storage, a relational database (e.g., PostgreSQL or MySQL) can be used for structured data storage.

- NoSQL databases like MongoDB or Cassandra can be employed for storing unstructured data or game events.

- Caching mechanisms (e.g., Redis) can be used to optimize read-heavy operations for ongoing games.

* **Historical Game Data**: For storing historical game data, consider using a relational database like PostgreSQL or MySQL. These databases are well-suited for structured data storage and complex queries.
* **Unstructured Data**: To store unstructured data or game events (e.g., player actions, scores, timestamps), NoSQL databases like MongoDB or Cassandra can be employed. These databases are designed for scalability and can handle high write loads.
* **Caching**: Implement caching mechanisms using Redis or similar technologies to optimize read-heavy operations for ongoing games. Caching can reduce the load on the database and improve response times.

**5. Content Delivery Network (CDN):**

- To efficiently deliver static assets like images and videos to users, a CDN like Cloudflare or Akamai can be utilized for faster content retrieval.

* Integrate a CDN like Cloudflare or Akamai to efficiently deliver static assets (e.g., images, videos) to users. CDNs have edge servers worldwide, reducing latency and improving content delivery.

**6. Authentication and Authorization:**

- Implement robust authentication and authorization mechanisms using OAuth 2.0 or JWT (JSON Web Tokens) for secure access control.

* **OAuth 2.0 or JWT:** Implement robust authentication and authorization mechanisms using OAuth 2.0 or JWT (JSON Web Tokens) to ensure secure access control. Users should be authenticated before accessing real-time game data.

**7. Scalability:**

- Auto-scaling capabilities should be implemented to handle sudden spikes in traffic. Cloud-based solutions allow for dynamic resource allocation.

- Use containerization (e.g., Docker) and orchestration (e.g., Kubernetes) for easy scaling and management of application instances.

* **Auto-scaling**: Set up auto-scaling capabilities for both the frontend and backend to handle sudden spikes in traffic. Cloud providers like AWS, Azure, and Google Cloud offer services for dynamic resource allocation.
* **Containerization and Orchestration**: Use containerization with Docker and orchestration with Kubernetes for easy scaling and management of application instances. This allows you to deploy and manage multiple containers efficiently.

**System Scaling Considerations:**

If the application needs to be accessed by 10,000 users simultaneously, several issues might arise:

**1. Server Load:** The backend servers may experience high CPU and memory usage, leading to performance degradation. To mitigate this, horizontal scaling with load balancers is crucial.

**2. Database Scalability:** Relational databases might struggle with concurrent read and write requests from a large number of users. Implementing database sharding or using NoSQL databases designed for scalability can help.

**3. Real-time Updates:** Scaling real-time data streaming can be challenging. The messaging system and WebSocket infrastructure should be able to handle the increased load without compromising latency.

**4. Cost Considerations:** Supporting 10,000 users simultaneously will likely increase infrastructure costs. Proper resource optimization and monitoring are essential to manage expenses efficiently.

**5. Security:** As user numbers grow, security becomes even more critical. Robust authentication and authorization mechanisms should be in place to prevent unauthorized access and protect user data.

In summary, this detailed system architecture leverages a combination of modern web technologies, cloud-based infrastructure, real-time communication tools, and robust database systems to create a scalable, reliable, and secure real-time basketball statistics application. Proper planning, monitoring, and optimization are crucial to ensure the system can handle the increased load when accessed by 10,000 users simultaneously.



