<think>  
Okay, let's tackle the question: What are the key components of system design?  
  
First, I need to recall the context provided. The user gave a bunch of material about system design interviews, key characteristics of distributed systems, and specific questions related to system design like single points of failure, data replication, serviceability, etc. The answer should come from that context.  
  
Looking at the context, the "System Design Basics" section lists several components. Let me check that again. The points mentioned there are:  
  
1. Different architectural pieces and how they work together.  
2. Utilizing these pieces with the right tradeoffs.  
3. The note about not scaling prematurely but planning ahead.  
  
Additionally, under "Key Characteristics of Distributed Systems," even though the text is cut off, the surrounding context includes aspects like scalability, fault tolerance, consistency, availability, partition tolerance (CAP theorem), load balancing, caching, data partitioning, replication, SQL vs NoSQL choices, etc.  
  
Also, in the questions provided earlier, the components related to avoiding single points of failure, data replication, serviceability (manageability), monitoring, and fault detection are important. The summary mentions architectural pieces and their interactions.  
  
Putting this together, the key components should include things like:  
  
- \*\*Architectural Components\*\*: Like load balancers, databases (SQL/NoSQL), caching layers, queues, proxies.  
- \*\*Scalability\*\*: Both vertical and horizontal scaling strategies.  
- \*\*Fault Tolerance/High Availability\*\*: Replication of data and services, avoiding single points of failure.  
- \*\*Load Balancing\*\*: Distributing traffic to prevent overloading any single component.  
- \*\*Caching\*\*: To reduce load and improve response times.  
- \*\*Data Partitioning/Replication\*\*: How data is stored and distributed across systems.  
- \*\*Monitoring & Alerting\*\*: Tracking performance and getting alerts on failures.  
- \*\*CAP Theorem Considerations\*\*: Trade-offs between Consistency, Availability, Partition tolerance.  
- \*\*Consistent Hashing\*\*: For data distribution in distributed systems.  
- \*\*Queues\*\*: For handling asynchronous processing and decoupling services.  
- \*\*Security\*\*: Though not explicitly mentioned in the provided text, it's a common component. Wait, the given context might not mention security, so maybe skip that unless it's implied.  
- \*\*Serviceability/Manageability\*\*: Ease of maintenance, diagnosing issues, updating systems.  
  
Wait, the context does mention serviceability as an important factor. The "Serviceability or Manageability" section talks about ease of repair, updates, and operation. So that's a key component.  
  
Also, the "System Design Interviews: A step by step guide" section lists some components like load balancing, replication, etc. The example about URL shortening might involve specific components, but perhaps the general components are more important here.  
  
Looking at the "System Design Basics" list again, the key components mentioned there include:  
  
- Consistent Hashing  
- CAP Theorem  
- Load Balancing  
- Caching  
- Data Partitioning  
- Indexes  
- Proxies  
- Queues  
- Replication  
- SQL vs NoSQL choices  
  
These are all listed as core building blocks. The key components of system design would encompass these elements. Also, fault tolerance, scalability, monitoring, and manageability as key considerations.  
  
So compiling all this, the answer should list the components like the architectural elements (databases, load balancers, etc.), design principles (CAP theorem, consistent hashing), strategies for scalability and fault tolerance (replication, partitioning), and operational aspects (monitoring, manageability). The exact terms from the context include the list under "core building blocks" in the System Design Basics section: Consistent Hashing, CAP Theorem, Load Balancing, Caching, Data Partitioning, Indexes, Proxies, Queues, Replication, and choosing between SQL vs NoSQL. Additionally, fault tolerance, high availability, monitoring, and serviceability are crucial.  
  
Wait, the user's question is about the key components, so they might be looking for the high-level components and concepts. The answer should list those components as per the provided context. Let me check again.  
  
The answer should list the key components from the given context. The context's "System Design Basics" section explicitly mentions the following as core building blocks for scalable systems:  
  
- Consistent Hashing  
- CAP Theorem  
- Load Balancing  
- Caching  
- Data Partitioning  
- Indexes  
- Proxies  
- Queues  
- Replication  
- SQL vs NoSQL  
  
These are listed as the core building blocks. Additionally, the key characteristics of distributed systems (even if the text was cut off, the CAP theorem is part of that) and other points like fault tolerance, scalability, monitoring, serviceability, etc., which are mentioned in other sections like the questions about SPOF, replication, and serviceability.  
  
Therefore, the key components would include these core building blocks plus considerations like fault tolerance, scalability strategies, monitoring/alerting, and manageability. The answer should list these components as per the context.  
</think>  
  
The key components of system design, based on the provided context, include:   
  
1. \*\*Architectural Components\*\*:\*\*   
 - Load balancers, databases (SQL/NoSQL), caching layers, queues, proxies, and indexes.   
  
2. \*\*Design Principles:\*\*   
 - \*\*CAP Theorem\*\* (trade-offs between Consistency, Availability, and Partition Tolerance).   
 - \*\*Consistent Hashing\*\* for data distribution.   
 - \*\*Data Partitioning\*\* and \*\*Replication\*\* strategies for scalability and fault tolerance.   
  
3. \*\*Fault Tolerance & High Availability:\*\*   
 - Avoiding \*\*single points of failure (SPOF)\*\* through redundancy (e.g., multiple replicas of data/services).   
 - \*\*Replication\*\* to ensure data/service availability even if some nodes fail.   
  
4. \*\*Scalability Strategies:\*\*   
 - Horizontal scaling (adding more nodes) and vertical scaling (upgrading hardware).   
  
5. \*\*Caching:\*\*   
 - Reducing latency and load on backend services by caching frequently accessed data.   
  
6. \*\*Monitoring & Alerting:\*\*   
 - Tools and processes to track system performance, detect failures, and alert on critical issues.   
  
7. \*\*Manageability (Serviceability):\*\*   
 - Ease of diagnosing issues, updating systems, and maintaining simplicity for smooth operations.   
  
8. \*\*Queues:\*\*   
 - Decoupling services and handling asynchronous processing (e.g., message queues).   
  
9. \*\*Data Storage Choices:\*\*   
 - Deciding between \*\*SQL\*\* (structured, ACID compliance) vs. \*\*NoSQL\*\* (flexible schema, horizontal scaling) based on requirements.   
  
10. \*\*Load Balancing:\*\*   
 - Distributing traffic to prevent overload and improve reliability.   
  
These components address scalability, fault tolerance, performance, and maintainability, as emphasized in the context.

Completed storing documents in rag\_documents\_ivf collection

Evaluating retrieval performance...

FLAT Index Results:

Collection: rag\_documents\_flat

Retrieval Time: 0.5996 seconds

Average Similarity: 0.4027

Number of documents retrieved: 5

HNSW Index Results:

Collection: rag\_documents\_hnsw

Retrieval Time: 0.6006 seconds

Average Similarity: 0.4027

Number of documents retrieved: 5

IVF Index Results:

Collection: rag\_documents\_ivf

Retrieval Time: 0.6088 seconds

Average Similarity: 0.4027

Number of documents retrieved: 5

Best performing index: flat

Best performing collection: rag\_documents\_flat

Reranking documents...

Generating response...