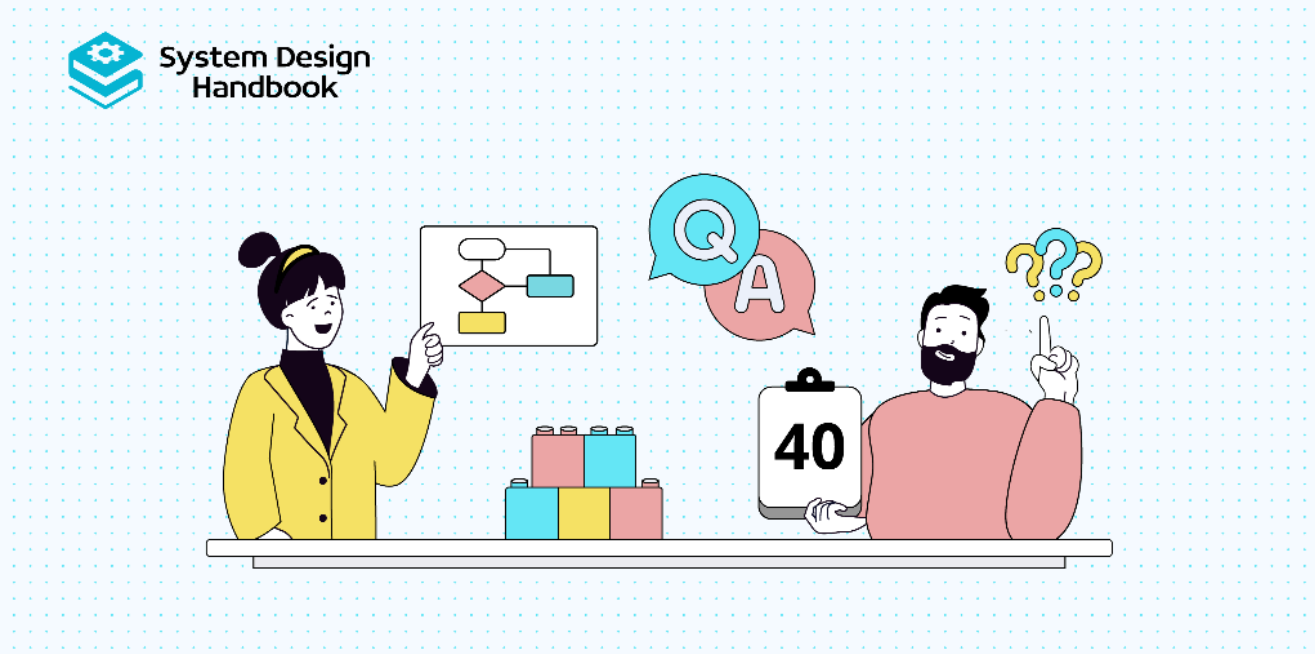
System Design Handbook
(https://www.systemdesignhandbook.com)

Top 40 System Design interview questions



Imagine you've aced the coding rounds and impressed the interviewers with your algorithms, and now you're facing the make-or-break moment of your tech interview—the System Design round.

Your task?

Design Twitter; or Netflix; or perhaps Uber, in the next 45 minutes.

System Design interviews (https://www.systemdesignhandbook.com/guides/system-design-interview/) have become the kingmaker in technical hiring, especially at tech giants that scale to multiple users. They're where theoretical knowledge meets real-world architecture, where "it works on my machine" transforms into "it works for a million machines."

Unlike coding interviews, where there's often a "right" answer, System Design is an art as much as a science. It's about making trade-offs, understanding business requirements, and architecting solutions that can evolve with time—skills that separate senior engineers from the pack.

In this comprehensive guide, I'll dive into 40 battle-tested System Design questions that have challenged candidates at FAANG and other leading tech companies. Whether preparing for your dream job interview or looking to level up your architectural thinking, I'll walk you through each problem with the same approach that has helped countless engineers succeed: breaking down complex systems into digestible components, understanding key requirements, and making informed design decisions.

Ready to transform from a coder to a system architect? Let's dive in and turn the following 40 tricky interview questions into opportunities for success.

Below are the 40 System Design interview questions:

1. Design a URL-shortening service
2. Design a distributed cache
3. Design a Google Maps system
4. Design a ride-sharing service
5. Design a recommendation service
6. Design a TripAdvisor system
7. Design a social media newsfeed service
8. Design a ticketing system
9. Design a payment gateway system
10. Design a video conference service
11. Design a distributed task scheduler
12. Design a collaborative editing service
13. Design a pub/sub system
14. Design a Pastebin service
15. Design a chess game
16. Design an e-commerce website

17. Design a Bluesky social network
18. Design a typeahead system
19. Design a Facebook search system
20. Design an electronic voting system
21. Design an ATM system
22. Design a multiplayer game
23. Design a parking lot system
24. Design an API system
25. Design a real-time messaging system
26. Design an online book review system
27. Design a content delivery (CDN) system
28. Design a forum-like system
29. Design a web crawler system
30. Design a ZooKeeper service
31. Design a distributed storage system like Bigtable
32. Design an API rate limiter for sites like Firebase
33. Design a DoorDash system
34. Design distributed locking service
35. Design an authentication system
36. Design location-based services like Yelp
37. Design notification service
38. Design a system to interview candidates
39. Design code deployment system
40. Design a stock exchange system

Let's explore the first design problem. I will briefly discuss the key steps to address most of the common questions, guiding you through the process of crafting a scalable solution.

1. Design a URL-shortening service

Problem statement: Design a scalable and distributed system for shortening long URLs. The system should generate unique short URLs from long URLs and allow users to retrieve the original URLs by accessing the shortened version.

We scope the problem by identifying the following requirements:

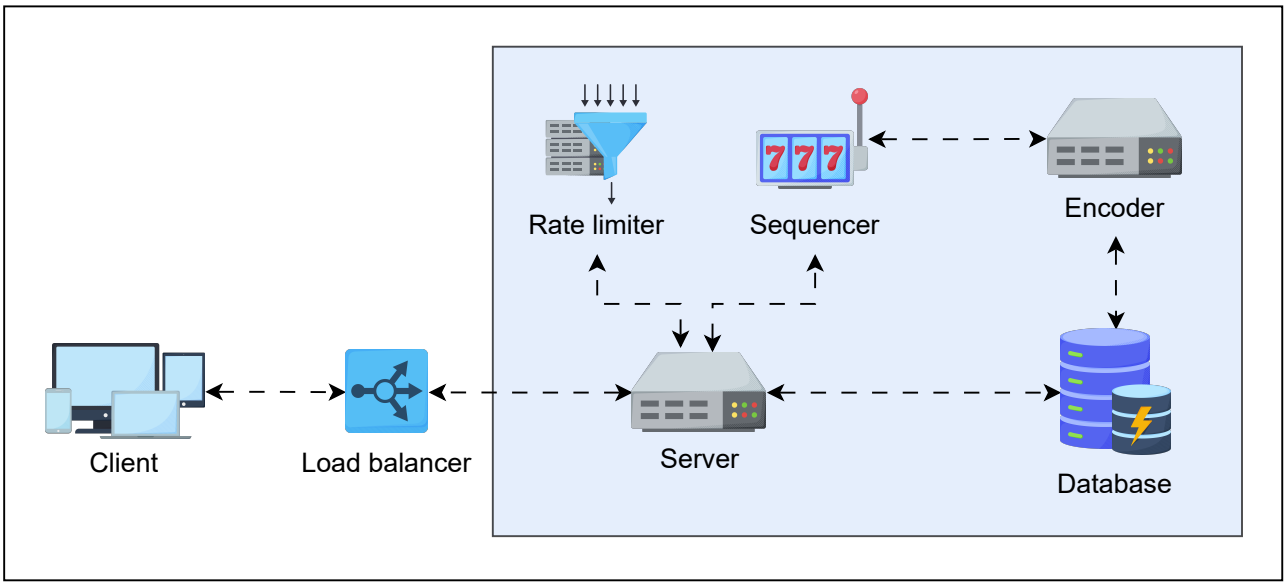
- **Functional requirements:**
  - **URL generation:** Generate a unique short URL for each long URL.
  - **URL storage:** Store the mapping between long and short URLs.
  - **Redirection to original URL:** Redirect users from the short to the original URL.
  - **Customization of URLs:** Allow users to customize the shortened URL.
  - **Update and delete URLs:** Enable users to update or delete shortened URLs.
- **Nonfunctional requirements:**
  - **Scalability:** Handle a high volume of URL shortening and redirection requests.
  - **Availability:** Ensure the system remains accessible with minimal downtime.
  - **Readability**: Ensure short URLs are readable and user-friendly.
  - **Low latency:** Minimize response times during URL shortening and redirection.
  - **Unpredictability in URL generation:** Ensure random and unique URL generation to avoid collision.

> **Note:** To better understand key aspects like scalability, availability, and performance in System Design, you might be interested in checking out the Nonfunctional Requirements for System Design Interviews (https://www.educative.io/blog/nonfunctional-requirements-for-system-design-interviews).

Now, let's look at how the system operates:

System workflow

In a URL-shortening service, when a user submits a long URL, the load balancer routes the request to a server, where the rate limiter checks if the user is within their request limits. If the request is valid, the server generates a unique ID using the sequencer, and the encoder converts this ID into a short alphanumeric string. The short URL and the corresponding original long URL are stored in the database. The system uses a cache to improve response time for frequently accessed URLs. When a user accesses a short URL, the system first checks the cache. It queries the database to retrieve the original URL and then sends it to the user if necessary.



The high-level design of URL shortening service

Test your knowledge!

- What mechanisms are in place to prevent and resolve collisions in auto-generated URLs?

Consider exploring the TinyURL System Design (https://www.educative.io/courses/grokking-the-system-design-interview/system-design-tinyurl) to answer such questions and gain a thorough understanding of the URL shortening service.

2. Design a distributed cache

Problem statement: Design a distributed caching system that ensures fast, scalable, and reliable data retrieval across multiple servers.

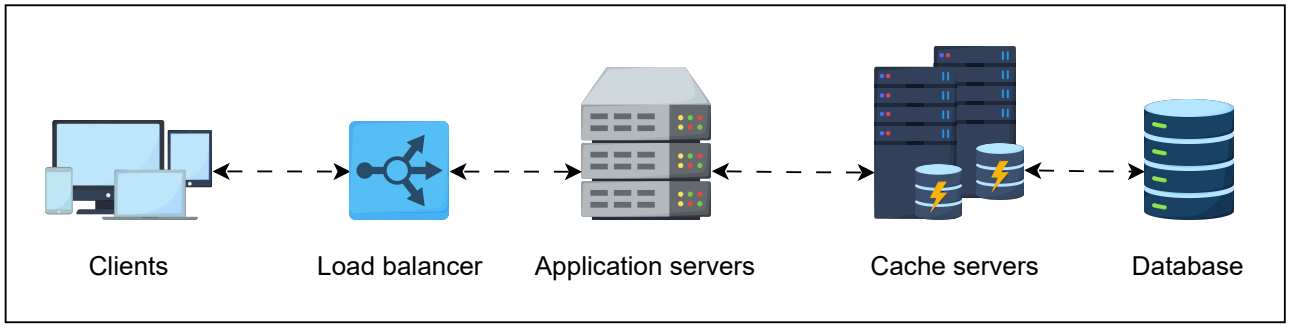
Let's consider the following requirements to design a distributed cache:

- **Functional requirements:**
  - **Insert data:** Allow data to be written into a cache.
  - **Retrieve data:** Provide fast data retrieval from the cache.
  - **Data partitioning:** Distribute data across multiple cache nodes.
  - **Cache eviction:** Implement cache eviction policies to remove stale data.
- **Nonfunctional requirements:**
  - **Scalability:** Support increasing data volumes and cache nodes without performance degradation.
  - **Consistency:** Ensure clients receive the most up-to-date data.
  - **Low latency:** Minimize delays in data retrieval.
  - **High availability:** Maintain access to cached data even during node failures.

Now, let's look at how the system operates:

System workflow

In a distributed cache system, data is partitioned across multiple cache nodes using a consistent hashing algorithm (https://grokkingthesystemdesign.com/blog/consistent-hashing-for-system-design/) to enhance access speed. The application server determines the appropriate cache node when a client requests data. If the data is found in the cache, it is returned immediately to ensure a rapid response. In case of a cache miss, the system fetches the data from the primary data store, stores it in the relevant cache node, and serves it to the client. An eviction policy manages cache storage by removing unused or expired data to maintain efficiency.



The high-level design of a distributed cache

> **Note:** You can explore the detailed System Design of a distributed cache (https://www.educative.io/courses/grokking-the-system-design-interview/system-design-the-distributed-cache) for an in-depth understanding.

3. Design a Google Maps System

Problem statement: Design a scalable system that provides real-time navigation, location search, route planning between two or more points, and multiple modes of travel.

Let's consider the following requirements to build the Google Maps system:

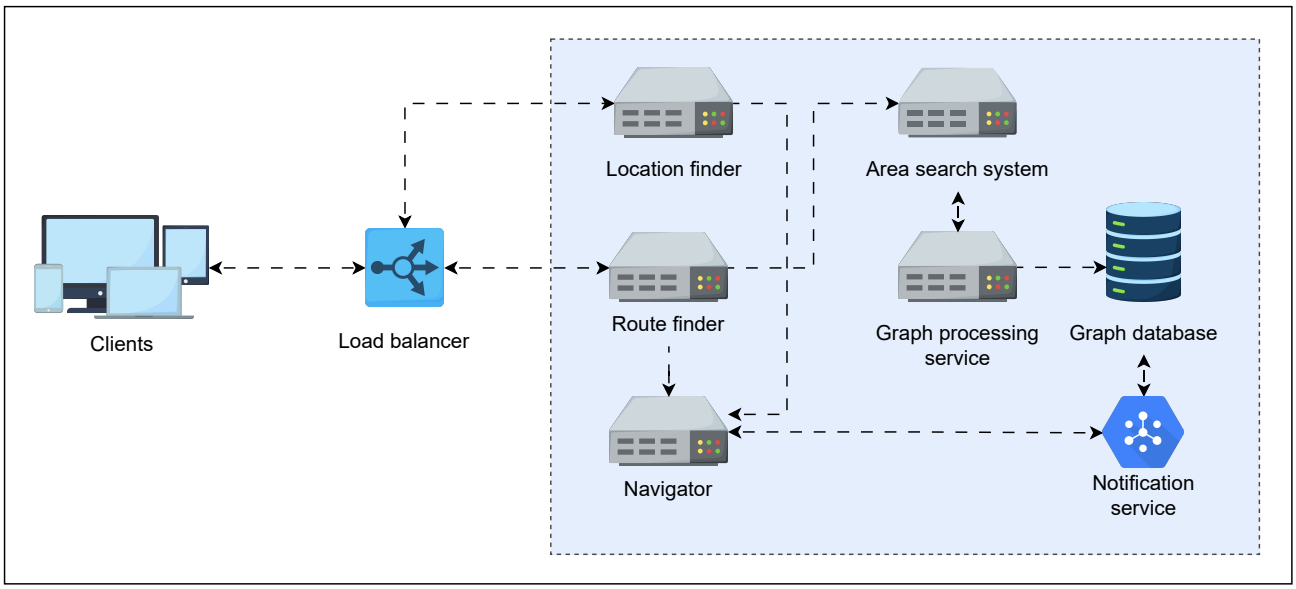
- **Functional requirements:**
  - **Real-time navigation:** Provide turn-by-turn navigation with live updates.
  - **Location search:** Allows users to search for locations or points of interest.
  - **Route finder:** Calculate optimal routes between locations.
  - **Route planning:** Suggest multiple routes based on travel mode.
  - **Real-time notifications:** Provide live traffic updates, delays, or road closures.
- **Nonfunctional requirements:**
  - **Scalability:** Handle millions of concurrent users and queries.
  - **Reliability:** Ensure continuous service availability.
  - **Accuracy:** Deliver precise routing and location details.
  - **Low latency:** Provide real-time results for user queries.

Now, let's look at how the system operates:

System workflow

In the Google Maps system, the client requests are routed through a load balancer to the appropriate service. The route finder service calculates optimal paths using real-time traffic data and historical trends, relying on the graph processing service to analyze the road network stored in the graph database. The location finder service identifies the user's location or resolves searched

coordinates and updates the user with the suggested path via notification service. In contrast, the area search system queries the graph database and third-party road data sources to locate nearby points of interest.



A high-level design of Google Maps

> **Note:** For more information, refer to the detailed design of a Google Maps (https://www.educative.io/courses/grokking-the-system-design-interview/system-design-google-maps).

4. Design a ride-sharing service

Problem statement: Design a system for a ride-sharing service that efficiently matches passengers with drivers, tracks real-time locations, handles payment securely, and ensures a seamless user experience.

Let's consider the following requirements to design a ride-sharing service:

- **Functional requirements:**
  - **Location tracking:** Monitor real-time locations of passengers and drivers.
  - **Request a ride:** Allow passengers to request a ride.
  - **Show nearby drivers:** Display available drivers near the passenger.
  - **Calculate and notify ETA:** Provide estimated time of arrival (ETA) based on vehicle type.

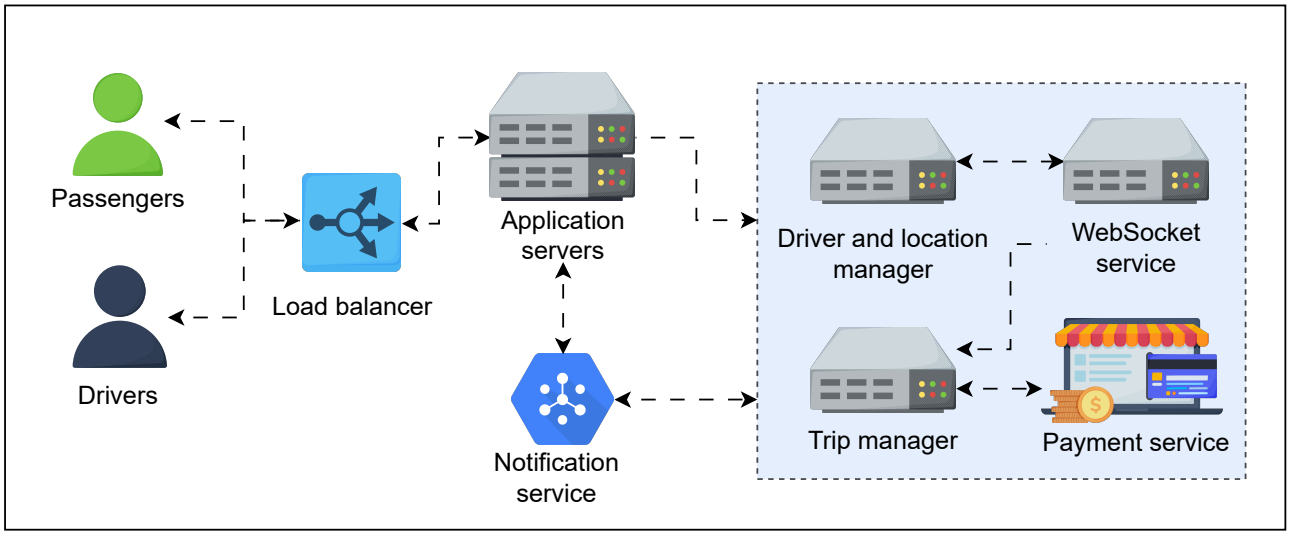
- **Trip process:** Manage ride confirmations, updates, and status changes.
- **Payment:** Handle secure and efficient payment processing.

- **Nonfunctional requirements:**
  - **Availability:** Ensure high uptime for reliable service.
  - **Security:** Protect user data and payment information.
  - **Reliability:** Accurately match drivers and passengers with minimal errors.
  - **Consistency:** Ensure correct and synchronized data for passengers and drivers.

Now, let's look at how the system operates:

System workflow

In a ride-sharing system, when a user requests a ride, the request is routed through a load balancer to the application server, where the trip manager calculates the ETA and sends it to the passenger. The driver and location manager use a matching algorithm to identify and notify nearby drivers via the notification service (i.e., pub/sub). Once a driver accepts the ride, trip details are shared with both parties. During the ride, the driver's location is updated in real time using GPS tracking and a notification service to keep both parties informed. Upon trip completion, the trip manager securely processes the payment through a payment service.



A high-level design of a ride-sharing service

To explore a ride-sharing system in detail, check out the design of Uber's backend (https://www.educative.io/blog/uber-backend-system-design).

5. Design a recommendation service

Problem statement: Design a recommendation engine that provides users with personalized and real-time suggestions based on their behavior, preferences, and past interactions.

Let's consider the following requirements to scope the design problem:

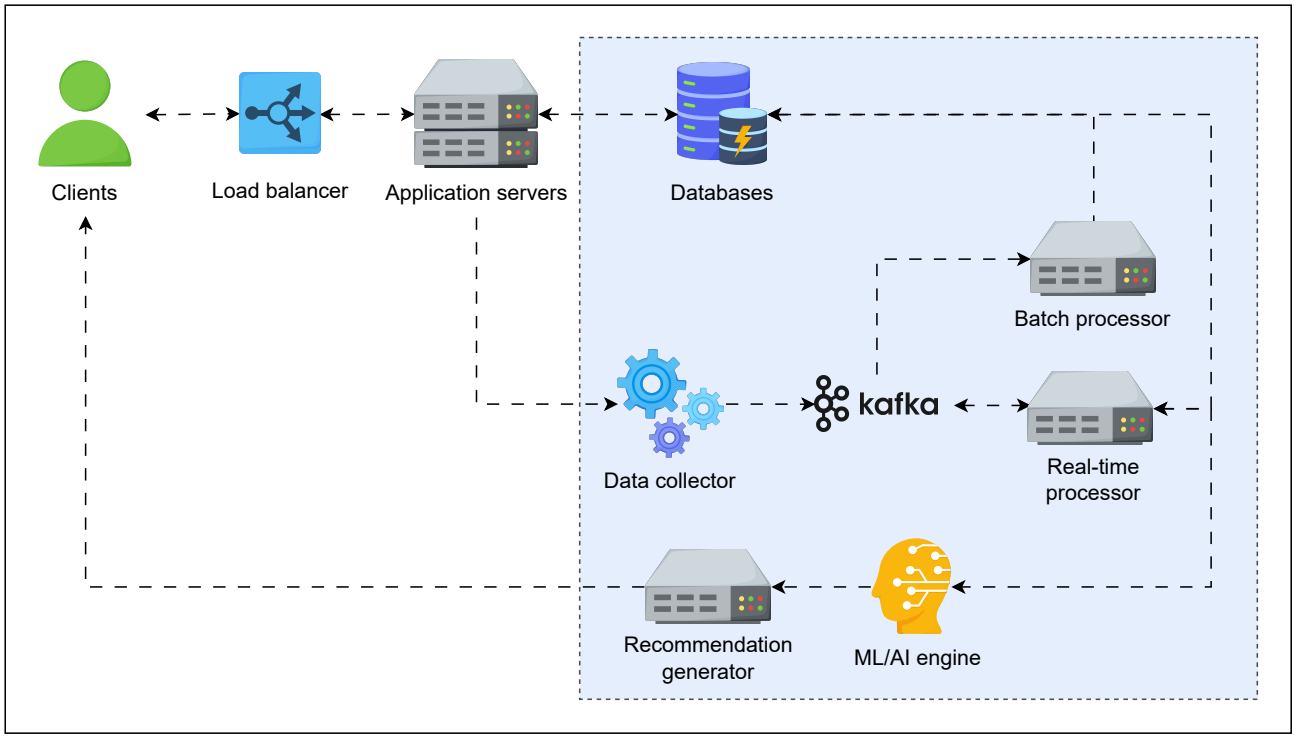
- **Functional requirements:**
  - **Offline processing:** Perform in-depth offline analysis to refine algorithms and enhance suggestions.
  - **Personalized recommendations:** Provide tailored suggestions based on user behavior.
  - **Update of recommendations:** Adjust recommendations dynamically with new interactions.
  - **Search and browse:** Enable users to explore content or products.
- **Nonfunctional requirements:**
  - **Privacy:** Protect user data from unauthorized access.
  - **Scalability:** Handle high volumes of users and interactions.
  - **Security:** Safeguards sensitive information during processing and storage.
  - **Low latency:** Ensure quick delivery of recommendations.

Now, let's look at how the system operates:

System workflow

The recommendation service operates in two stages:

- **Data collection and processing:** The data collector gathers user interactions and logs them into Kafka for processing. There are two types of data processing:
  - **Real-time processor:** It analyzes streaming data for immediate recommendation updates.
  - **Batch processor:** It performs offline analysis to refine algorithms and improve accuracy.
- **Recommendation generation:** Processed data is used by an ML/AI engine with algorithms like collaborative filtering, content-based filtering, and hybrid methods to deliver personalized suggestions that are updated dynamically based on user behavior.



A high-level design of a recommendation service

6. Design a TripAdvisor system

Problem statement: Design a system for travel recommendations and booking that allows users to search for and book hotels, restaurants, flights, and activities. The system should feature user-generated reviews, photos, and ratings to assist travelers in making well-informed decisions.

Some of the core features of TripAdvisor are as follows:

- Search and discovery
- Recommendation engine
- Booking system
- Reviews and ratings

Test your knowledge!

- How will the system store user-generated content like reviews, photos, and ratings?
- Should reviews have a time decay factor to prioritize newer content?
- How will third-party APIs for booking services be integrated and managed?

To answer more such interesting questions, explore the design of a recommendation system (https://www.educative.io/blog/netflix-system-design-interview-questions#3-Design-a-personalized-recommendation-system).

7. Design a social media newsfeed service

Problem statement: Create a social-media newsfeed system that delivers users personalized, real-time updates.

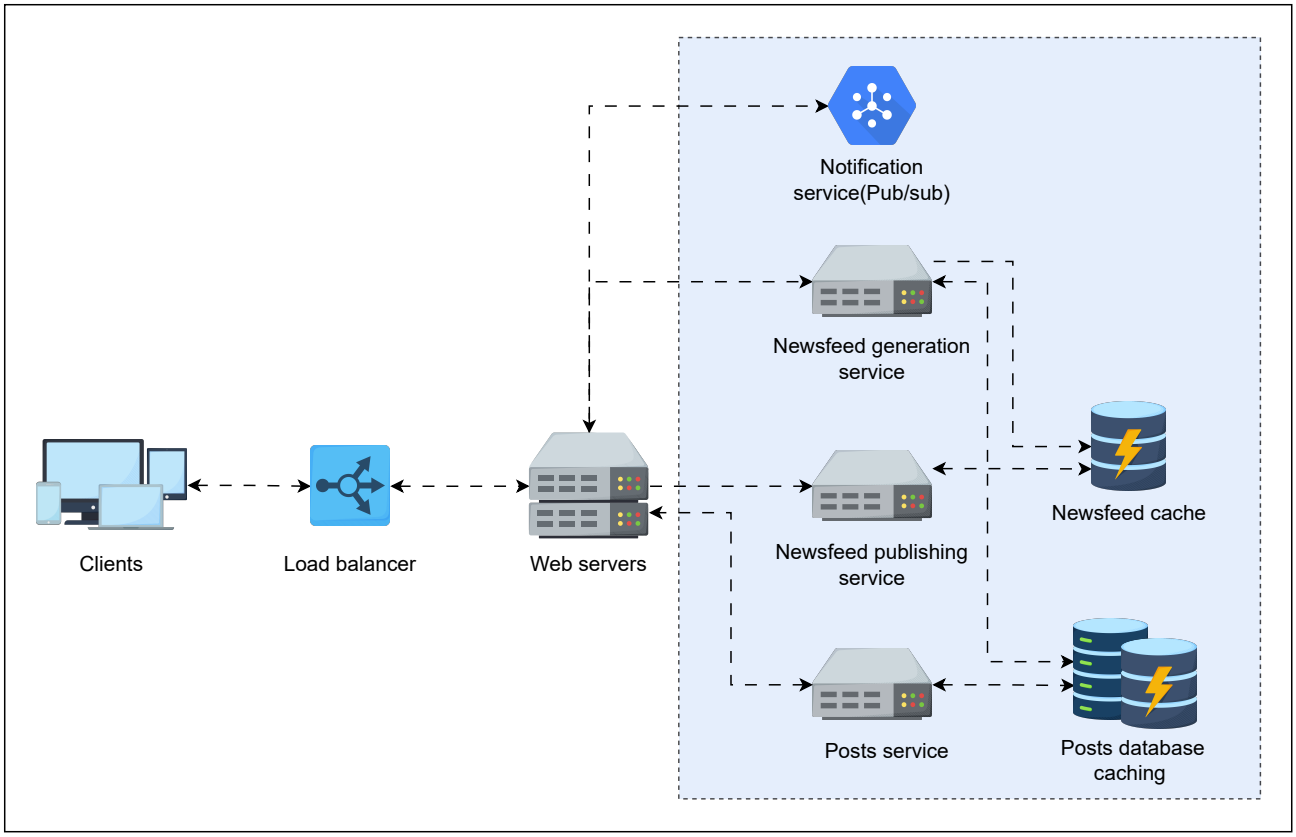
For designing a social media newsfeed service, well consider the following requirements:

- **Functional requirements:**
  - **Newsfeed generation:** Automatically generate personalized feeds based on user interactions and followers.
  - **Newsfeed content:** Include posts, status updates, images, and multimedia content from friends or followers.
  - **Newsfeed display:** Delivers the generated newsfeed to user's timelines.
- **Nonfunctional requirements:**
  - **Availability:** Keep the system available for users at all times.
  - **Low latency:** Ensure real-time content delivery with minimal delay.
  - **Scalability:** The system should support millions of users and high-frequency updates.
  - **Fault tolerance:** Ensure the system can handle failures without affecting performance.

Now, let's look at how the system operates:

System workflow

In a social media newsfeed system, user actions are routed through a load balancer to web servers for authentication. When a user posts, notification (i.e., pub/sub) and newsfeed generation services spring into action, aggregating, ranking, and caching personalized feeds. On request, the newsfeed publishing service retrieves the feed from the newsfeed cache, seamlessly adds multimedia from blob storage if needed, and delivers it to the user.



A high-level design of a newsfeed service

> **Note:** To understand the intricacies of ranking algorithms and real-time updates, check out the detailed design of a newsfeed service (https://www.educative.io/courses/grokking-the-system-design-interview/system-design-newsfeed-system), which provides a step-by-step breakdown.

8. Design a ticketing system

Problem statement: Design a ticketing system that allows users to book tickets for events, movies, or transportation, manage their bookings, and receive updates on their purchases.

Let's consider the following requirements for designing a ticketing system:

- **Functional requirements**
  - **Registration:** Users can sign up, log in, and manage their profiles.
  - **Even listings:** Display a catalog of events, movies, or transportation schedules.
  - **Seat selection:** Allow users to view and select available seats in real time.
  - **Ticket booking:** Enable users to reserve and purchase tickets.

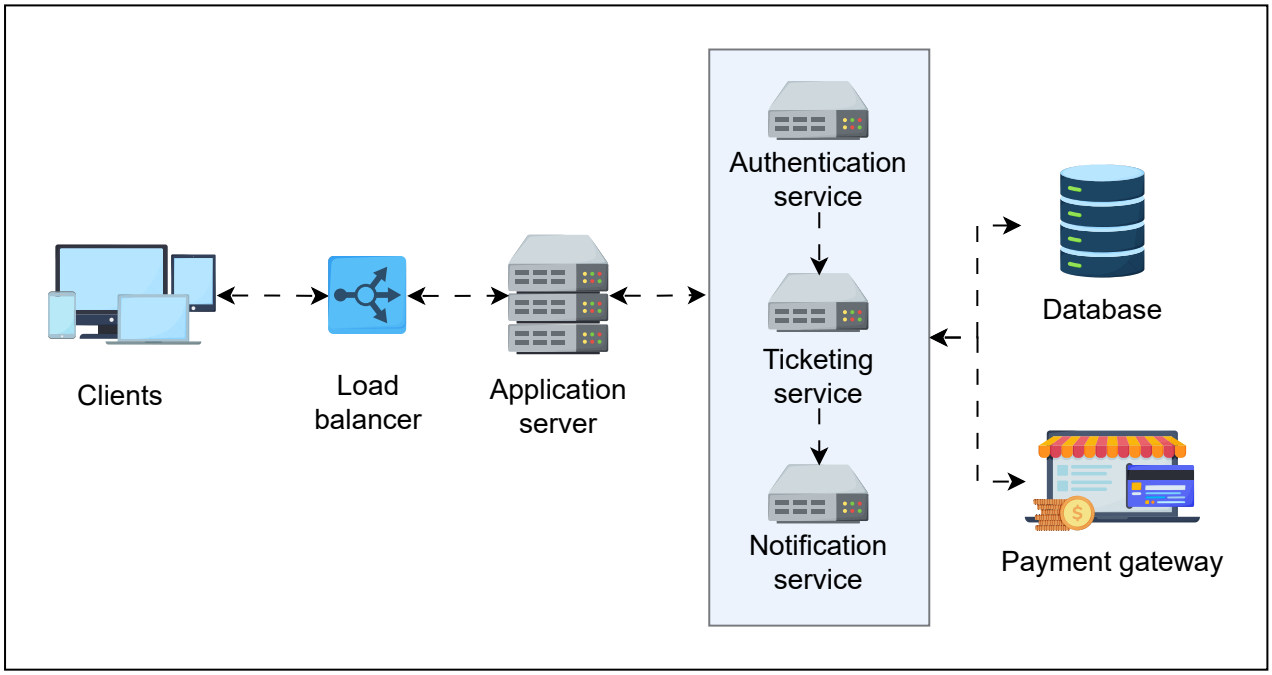
- **Payment:** Support multiple payment methods for secure transactions.

- **Nonfunctional requirements**
  - **Scalability:** Handle high traffic during peak times, such as popular event launches.
  - **Security:** Protect user data and payment information with encryption and secure authentication.
  - **Fault tolerance:** Recover from failures (such as payment gateway issues) without disrupting user activity.

Now, let's look at how the system operates:

System workflow

In a ticketing system, users log in and select an event, movie, or transportation schedule. They view the available seats in real time, choose their preferred seats, and proceed to checkout, where they provide payment details and confirm the booking. Once payment is processed, the system generates a ticket, updates seat availability, and sends a confirmation notification to the user.



A high-level design of a ticketing system

Test your knowledge!

- How would you design a ticketing system to handle high concurrency during the release of a popular event and real-time seat availability updates?

- How will the system prevent double bookings on a single seat?

9. Design a payment gateway system

Problem statement: Design a secure and reliable payment gateway system that supports online transactions, handles millions of users, and ensures seamless payment processing.

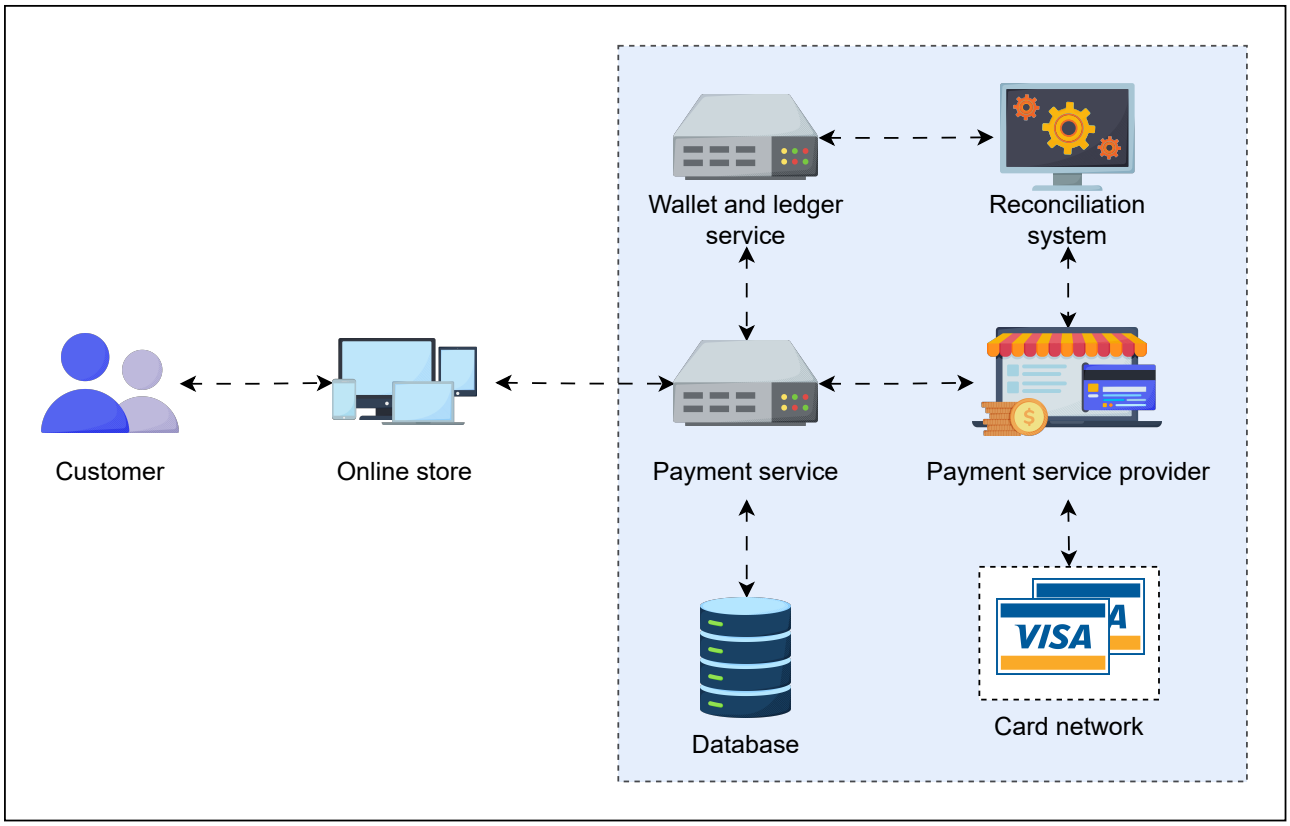
Let's consider the following requirements for designing a payment gateway:

- **Functional requirements:**
  - **User registration and authentication:** Allow merchant and customer to register and log in securely.
  - **Payment processing:** Handle card and online transactions, including security checks.
  - **Mobile accessibility:** Provide easy integrations with mobile platforms.
  - **Transaction history:** Maintain detailed user records.
  - **Balance management:** Enable merchants to view and manage earnings.
- **Nonfunctional requirements:**
  - **Performance:** Handle millions of concurrent users and transactions.
  - **Availability:** Ensure the system is always operational.
  - **Reliability:** Guarantee accurate payment processing.
  - **Data integrity and security:** Protect sensitive data through encryption and secure protocols.

Now, let's look at how the system operates:

System workflow

In a payment gateway system, the payment service processes customer payment details performs security checks and forwards the request to a payment service provider (PSP) for validation and authorization via card network APIs. Upon a successful transaction, the wallet and the ledger update merchant accounts, track revenue, and handle multi-seller transactions. The reconciliation system compares PSP, wallet, and ledger transaction records to ensure data consistency, resolve discrepancies, and provide accurate financial reporting across all systems.



A high-level design of a secure payment gateway system

> **Note:** You might be interested in exploring the Stripe API design (https://www.educative.io/courses/grokking-the-api-design-interview/stripe-api-design-decisions) for an in-depth understanding.

10. Design a video conference service

Problem statement: Create a system for hosting high-quality video meetings, supporting real-time communication, meeting controls, and varying network conditions.

Let's consider the following requirements for a video conference service:

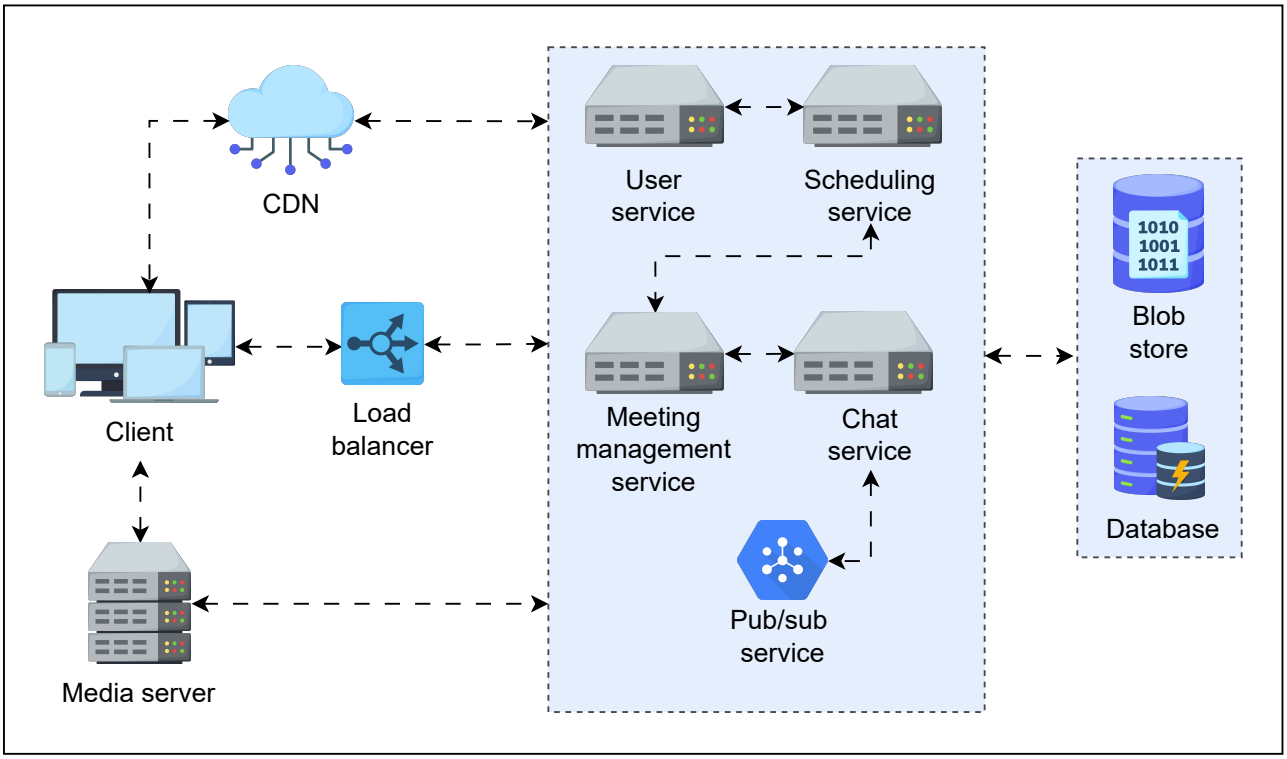
- **Functional requirements:**
  - **Schedule a meeting:** Allow users to schedule and organize meetings.
  - **Real-time chat:** Support real-time text-based interaction during meetings.
  - **Manage participants:** Handle roles, permissions, and joining/leaving participants.
  - **Stream data (audio and video):** Provide high-quality audio and video streaming.
  - **Record meeting:** Allow meetings to be recorded for late use.

- **Nonfunctional requirements**
  - **Scalability:** Handle increasing users and simultaneous meetings.
  - **Availability:** Ensure minimal downtime for a seamless experience.
  - **Low latency:** Ensure real-time communication with minimal downtime.
  - **Security for private meetings:** Encrypt data for secure and private meetings.

Now, let's look at how the system operates:

System workflow

In the video conference system, user requests pass through a load balancer to backend services. The user service handles user authentication, profile management, and user preferences. The scheduling service organizes meeting details, while the meeting management service controls participant roles, permissions, and access. A media server encodes and decodes real-time audio and video for media streaming, while a dedicated screen-sharing module allows participants to share their screens. With data encryption, the chat service enables text-based communication to ensure secure chats and protect user information. The pub/sub service ensures real-time message delivery and notifications, while the blob store handles the storage of multimedia content like recorded meetings and shared files. Finally, the database stores user data, meeting logs, and user's event-related activity information for accurate tracking and reporting.



A high-level design of a video conference service

> **Note:** Gain a deeper understanding of low-latency streaming and media routing by exploring the Zoom API design chapter (https://www.educative.io/courses/grokking-the-api-design-interview/requirements-of-the-zoom-api).

11. Design a distributed task scheduler

Problem statement: Design a distributed task scheduler that should manage and schedule tasks across multiple servers. It should also ensure task execution at a specified time.

Let's consider the following requirements for designing a distributed task scheduler:

- **Functional requirements:**
  - **Scheduling:** Allow users to schedule tasks at specified times.
  - **Execution:** Execute tasks on available servers based on scheduling criteria.
  - **Rescheduling:** Reschedule tasks in case of failure or server unavailability.
  - **Prioritization:** Support prioritizing critical tasks over less important ones.
- **Nonfunctional requirements:**
  - **Scalability:** Support the addition of nodes to handle increasing task loads.
  - **Availability:** Ensure continuous operation of the task scheduler with minimal downtime.
  - **Latency:** Deliver real-time scheduling and execution with minimal delays.
  - **Consistency:** Maintain the consistency of task execution, even in a distributed setup.

Test your knowledge!

- How do you measure the task execution rate in your distributed task scheduler, ensuring tasks are evenly distributed across nodes without overloading any single server?
- How do we handle the issue of task prioritization?

For more such interesting questions and their answer, see the detailed design of the distributed task scheduler (https://www.educative.io/courses/grokking-the-system-design-interview/system-design-the-distributed-task-scheduler).

12. Design a collaborative editing service

Problem statement: Design an editing system like Google Docs, enabling multiple users to simultaneously edit documents remotely with real-time updates and the ability to manage document histories.

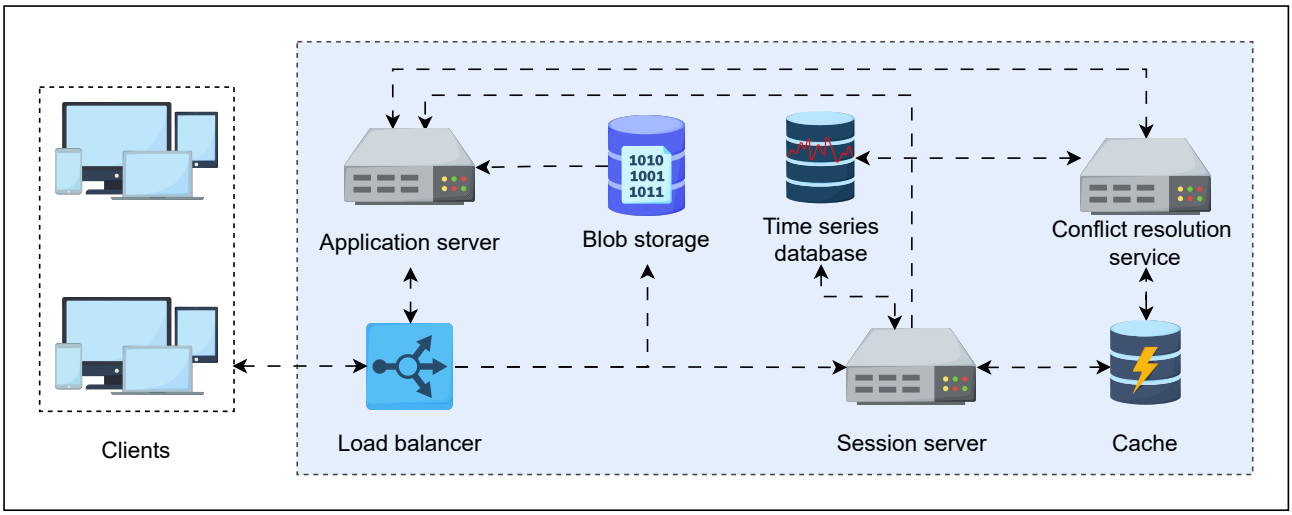
Let's consider the following requirements for designing a collaborative editing service:

- **Functional requirements**
  - **Collaboration:** Enable multiple users to edit the document simultaneously.
  - **Edit overlap:** Efficiently manage and display simultaneous changes made by users.
  - **Autocomplete and grammatical suggestions:** Provide real-time text predictions and corrections.
  - **History and view count:** Track document revisions and views for collaboration transparently.
  - **Manage documents:** Allow users to create, edit, and delete documents.
- **Nonfunctional requirements**
  - **Consistency:** Users can see the same document state in real time.
  - **Availability:** Ensure the document is always accessible to logged-in users.
  - **Low latency:** Ensure real-time updates and minimal delays during document edits.

Now, let's look at how the system operates:

System workflow

In the collaborative editing service, when a user makes an edit, the request is sent to an operations queue, where concurrent edits are managed. The conflict resolution service merges change in real time, ensuring seamless collaboration. Document changes are stored in a time series database, which tracks revisions over time, while Blob storage holds larger media files like images and videos. Each change is time-stamped for accurate revision history.



A high-level design of collaborative editing service like Google Docs

> **Note:** For more details you can see the detailed design of Google Docs (https://www.educative.io/courses/grokking-the-system-design-interview/system-design-google-docs).

13. Design a pub/sub system

Problem statement: Design a scalable and distributed pub/sub system that supports high-throughput messaging, ensuring reliable delivery.

Let's address the following requirements for designing a pub/sub system:

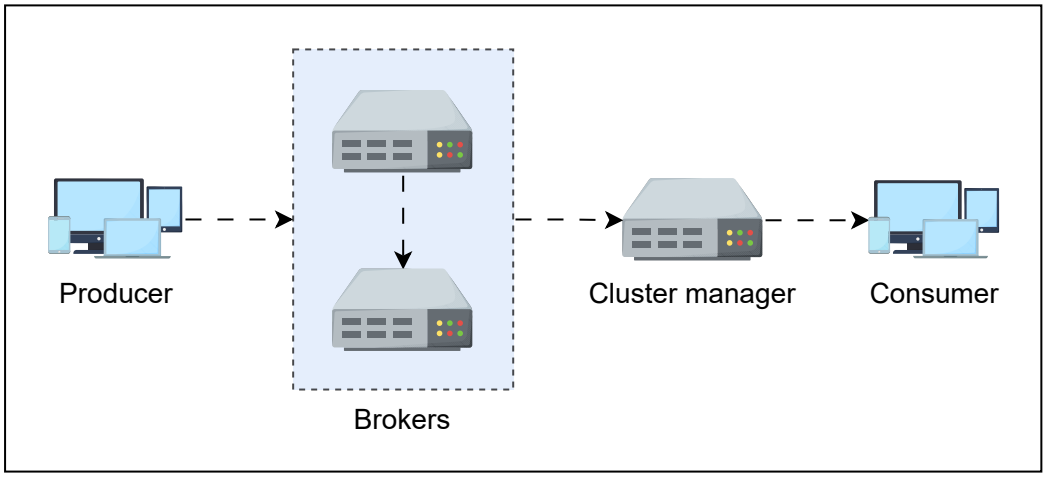
- **Functional requirements:**
  - **Topic creation:** The system should allow publishers to create logical channels for message organization.
  - **Subscribe to topics:** Let consumers subscribe to topics of interest.
  - **Write and read messages:** Enable publishers to send messages and consumers to fetch messages from their subscribed topics.
  - **Delete messages:** Allow deletion of messages beyond the retention period or upon request.
  - **Retention policy:** Support configurable retention times for storing messages.
- **Nonfunctional requirements:**
  - **Availability:** Ensure the system remains operational during failures.

- **Scalability:** Handle increased message volumes and consumer counts.
- **Concurrency management:** Efficiently manage simultaneous reads and writes.
- **Durability:** Safeguard message persistence against system crashes.
- **Fault tolerance:** Recover seamlessly from broker or network failures.

Now, let's look at how the system operates:

System workflow

In a publisher-subscriber system, producers send messages to brokers under specific topics. These brokers store and replicate the messages across the cluster to ensure durability. Subscribers register with the broker to express interest in particular topics, facilitating their subscription. Consumers retrieve messages from the brokers based on their subscriptions, tracking offsets to ensure reliable message delivery. The cluster manager constantly monitors broker health, and in the event of a failure, it automatically spins up a new broker and redistributes the load, ensuring continuous system availability and reliability.



The high-level design of a pub/sub system

> **Note:** For an in-depth understanding, you can explore the detailed design of pub/sub. (https://www.educative.io/courses/grokking-the-system-design-interview/system-design-the-pub-sub-abstraction)

14. Design a Pastebin service

Problem statement: Design a text storage web application that allows users to paste text and access the text and code snippets publically or privately anytime.

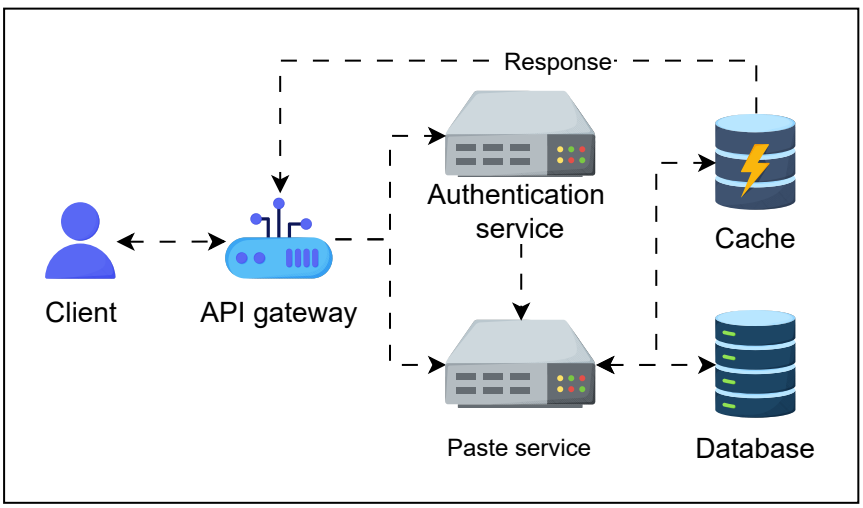
Let's consider the following requirements to design a Pastebin service:

- **Functional requirements**
  - **Create and edit:** The user can create, edit, and delete pasted text.
  - **Expiration time:** The user can set expiration time for the pasted text.
  - **Unique URL:** Create a unique URL for the pasted text, which can be accessed anywhere.
- **Nonfunctional requirements**
  - **Scalability:** Handle the large number of concurrent users.
  - **Availability:** The pasted text should be accessible all the time.
  - **Security:** Data security to protect confidential pastes.
  - **Searching:** Efficient search functionality for public pastes.

Now, let's look at how the system operates:

System workflow

The Pastebin workflow begins when a user requests to create or view a paste. The request is routed through the API gateway, which forwards it to the authentication service for access control. Once authenticated, the request reaches the paste service, which retrieves data from the cache if available or queries the database otherwise. The response is then sent back to the user via the API gateway.



A high-level design of a Pastebin service

Test your knowledge!

- How can Pastebin ensure that private pastes remain secure from unauthorized access, especially in cases where users accidentally share their private URLs or if malicious actors target the system?

We can use the following:

- **URL encryption/tokenization:** Use tokens in URLs that are not easily guessable, valid only for a short period, or require additional authentication to access.
- **Logging and monitoring:** Monitor and log all access attempts, especially for private pastes, to detect potential breaches or unauthorized access.

For more insights on designing large-scale, secure, and efficient systems, check out "Grokking the Modern System Design Interview (https://www.educative.io/courses/grokking-the-system-design-interview)."

15. Design a chess game

Problem statement: Design a chess game that allows two users to play against each other. The system must manage the game state and enforce chess rules in real time.

Let's consider the following requirements for designing a chess game:

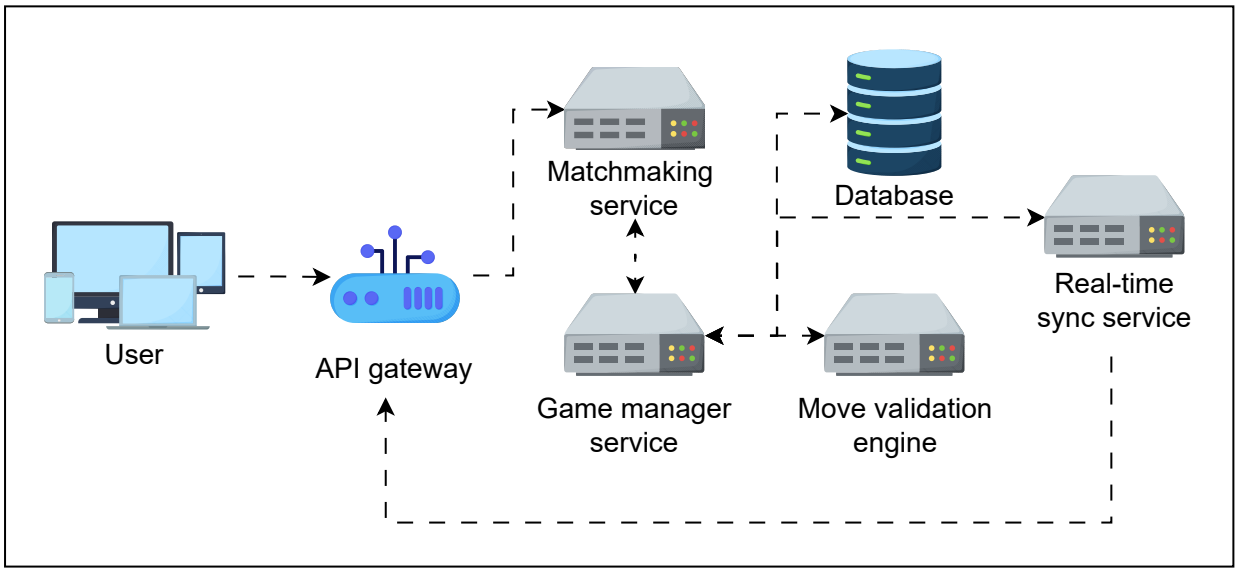
- **Functional requirements**
  - **Move validation:** Validate the legality of each move based on the chess rules.
  - **Game state persistence:** The game state must be saved, allowing players to resume the game anytime.
  - **User authentication:** Users should be able to sign in to track their game history or play anonymously.
  - **Matchmaking:** A feature to connect players with opponents of similar skill levels online.
- **Nonfunctional requirements:**
  - **Scalability:** Handle a large number of concurrent users.
  - **Availability:** The game should be available 24/7 to ensure players can access and play anytime.

- **Performance:** The game must have low latency in processing moves and updating the game board, providing a smooth and responsive experience.
- **Consistency:** Maintain game state consistency across different devices and players during real-time online matches.

Now, let's look at how the system operates:

System workflow

When a user logs in or plays anonymously in a chess game, their request passes through the API gateway to the matchmaking service, which pairs them with an opponent or sets up a new game. The game manager service initializes the chessboard and manages the game state as players make moves. The move validation engine validates each move, and if valid, the game manager service updates the board and stores the game state in the database. The real-time sync service ensures the updated game state is sent back to both players, enabling smooth gameplay.



A high-level design of chess game

Test your knowledge!

- How can we design a system that can handle real-time synchronization and consistency in a chess game played between users across different regions, ensuring minimal lag and no game state discrepancies, even in the event of network issues or server failures?

For more insights on handling real-time synchronization, performance, and fault tolerance in multiplayer systems, check out our chess API design (https://www.educative.io/courses/grokking-the-api-design-interview/chess-api-design-ai-mentor-beta).

16. Design an e-commerce website

Problem statement: Design an e-commerce website that allows users to browse, search, and purchase products, manage their shopping cart, and complete transactions securely.

Let's consider the following requirements to design an e-commerce website:

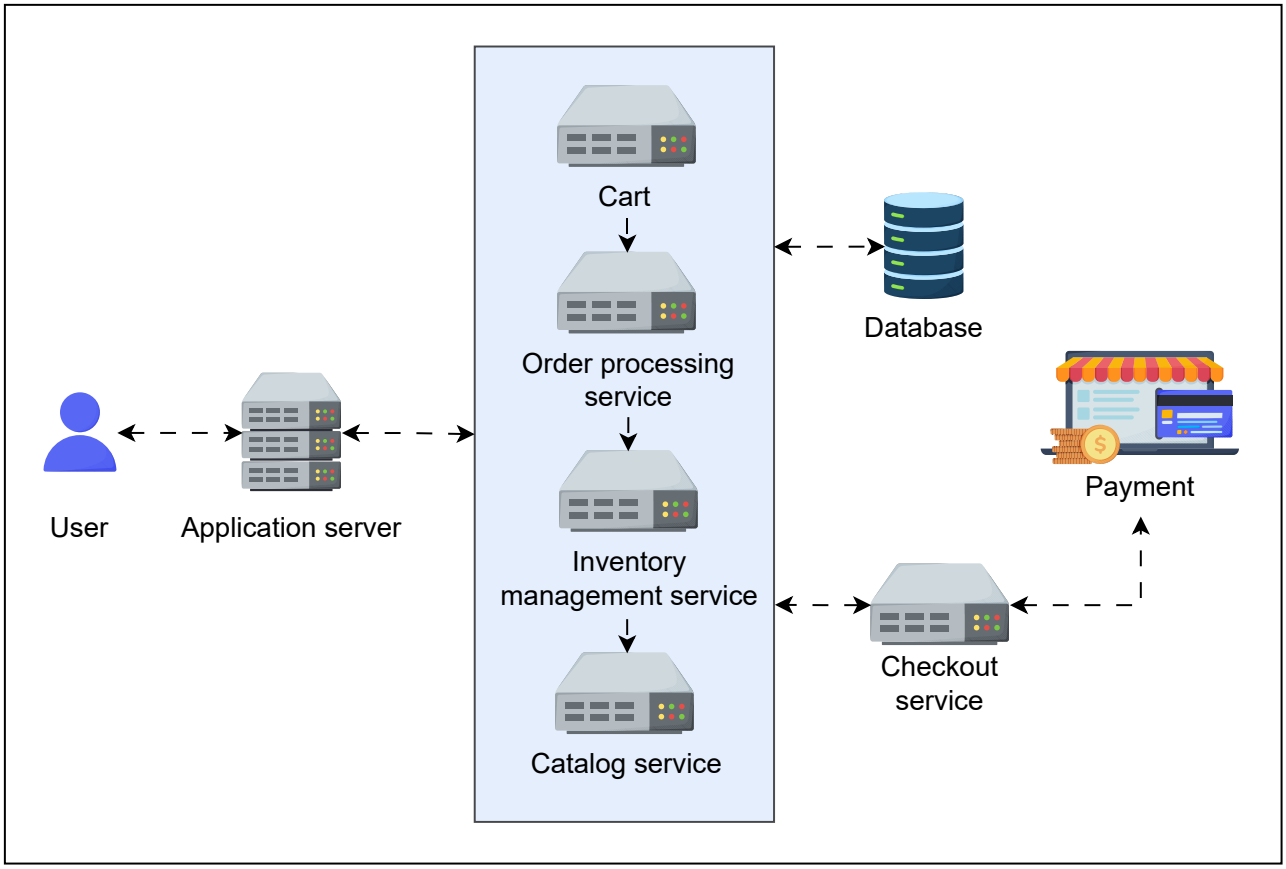
- **Functional requirements:**
  - **User authentication:** Allow users to sign up, log in, and manage their profiles.
  - **Product catalog:** Users should be able to browse and search for products.
  - **Cart:** Users can add, update, or remove items from the shopping cart.
  - **Checkout:** Handle billing information, delivery address, and payment.
- **Nonfunctional requirements:**
  - **Performance:** The website should load quickly to provide a smooth experience to customers.
  - **Security:** Implement strong security measures and payment processing to protect user data and transactions.
  - **Availability:** The website should be accessible anytime.
  - **Fault tolerance:** Recover from failures without affecting the user experience.

Now, let's look at how the system operates:

System workflow

In an e-commerce website, when a user starts browsing the product catalog, the catalog service handles their requests and retrieves product information from the database. Upon selecting a product, the user views its details, adds it to their shopping cart, and proceeds to checkout.

During checkout, the user provides shipping details, selects a payment method, and confirms the order, which the processing service processes. Once the purchase is completed, the inventory management service updates the stock in the database, reflecting real-time availability. The system then sends an order confirmation to the user and triggers the fulfillment process to initiate shipping.



A high-level design of an e-commerce service

Test your knowledge!

- How would you design a system that will handle sudden spikes in traffic, such as during a flash sale, while ensuring that transactions are processed securely and the user experience remains uninterrupted?

See the detailed discussion on designing an e-commerce platform (https://www.educative.io/courses/grokking-the-system-design-interview/ai-evaluation-of-building-blocks-in-e-commerce-platform).

17. Design a Bluesky social network system

Problem statement: Design a decentralized social network like Bluesky that enables users to create and share content while maintaining control over their data. The system should ensure scalability, interoperability, and user privacy while fostering an open ecosystem for innovation and reducing dependency on centralized platforms.

> **Note:** To dive deeper into the details of social networking, explore the Bluesky Social Network (https://www.educative.io/blog/bluesky-social-network-system-design) System Design.

18. Design a typeahead system

Problem statement: Design a typeahead system that predicts and suggests a list of words or phrases based on a user's input. The system should handle high scalability and provide results with minimal latency.

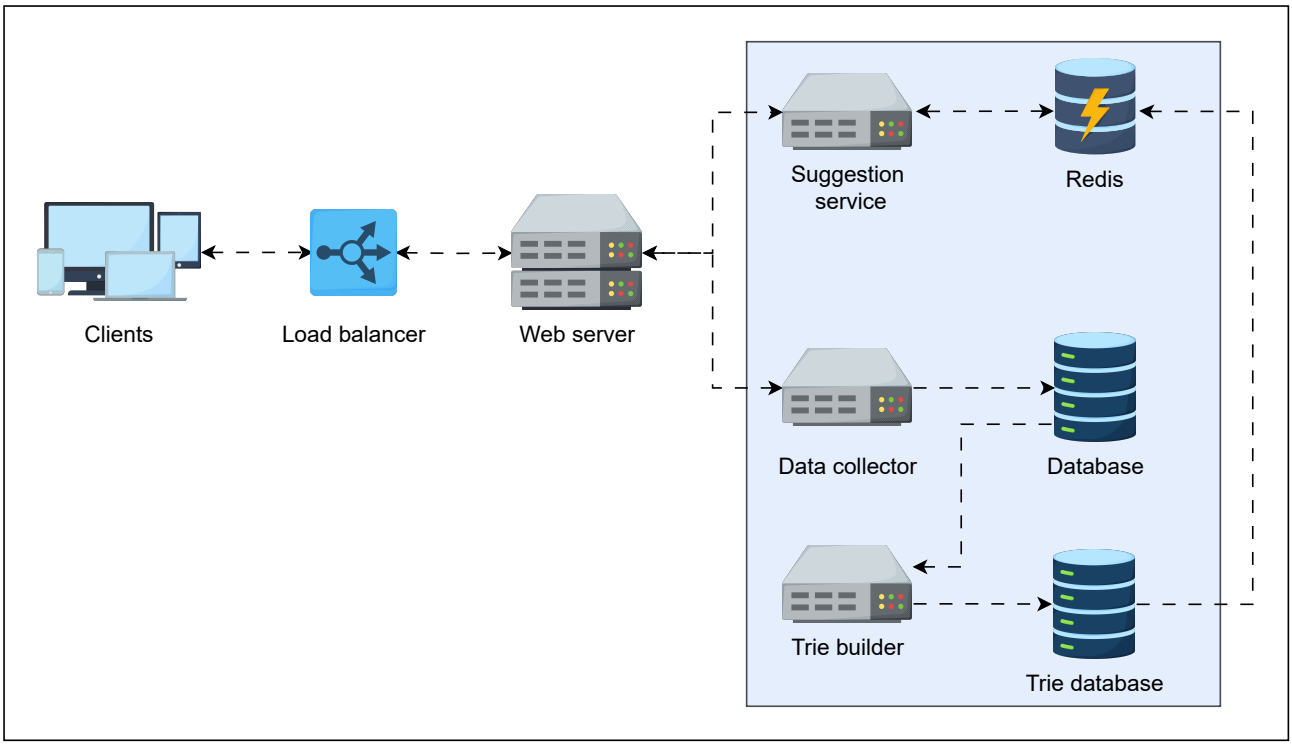
Let's start with the following requirements:

- **Functional requirements**
  - **Suggestions:** Show suggestions as the user types.
  - **Personalization:** Tailor suggestions based on user history or preferences.
  - **Ranked results:** Suggestions should be ranked based on relevance and popularity.
  - **Low latency**: Ensure suggestions are served in real-time, i.e., in a few milliseconds.
- **Nonfunctional requirements**
  - **Availability:** The system should have minimal downtime.
  - **Fault tolerance:** Handle server or network failures gracefully.
  - **Consistency:** Provide consistent suggestions even in a distributed setup.

The typeahead system will have the following workflow:

System workflow

The typeahead system uses a trie or prefix tree stored in memory (within the search service or a cache layer like Redis) for efficient string matching. It is combined with a search service to fetch and rank suggestions based on relevance, popularity, and personalization. A cache layer (e.g., Redis) handles frequently accessed queries for low latency, while a database stores metadata like query frequency and user preferences and updates to the Trie structure for persistence.



A high-level design of a typeahead system

Test your knowledge!

- How does the Typeahead system balance personalizations and global relevance?

For more information, see the detailed System Design of the typeahead suggestion system (https://www.educative.io/courses/grokking-the-system-design-interview/system-design-the-typeahead-suggestion-system).

19. Design a Facebook search system

Problem statement: Design a Facebook status search system that enables users to search for specific posts or updates on the platform. The system must process text, images, and multimedia content in user statuses and support retrieval based on keywords, hashtags, or user tags, ensuring fast and accurate results at scale.

Test your knowledge!

- How does the Facebook search system balance the real-time indexing of new posts with the need for fast search response times?

- How does Facebook handle high traffic and ensure the scalability of the search system?
- How does the system ensure accurate search results based on user queries and filters (filters, hashtags, etc.)?

To answer more such questions, check out the detailed design of the search system (https://www.educative.io/courses/grokking-the-system-design-interview/system-design-the-distributed-search) at Educative.

20. Design an electronic voting system

Problem statement: Design an electronic voting system that enables secure digital ballot casting. The system must ensure voter authentication, vote privacy, tamper-proof data storage, double voting prevention, and voter anonymity maintenance throughout the process.

Let's consider the following requirements for designing an electronic voting system:

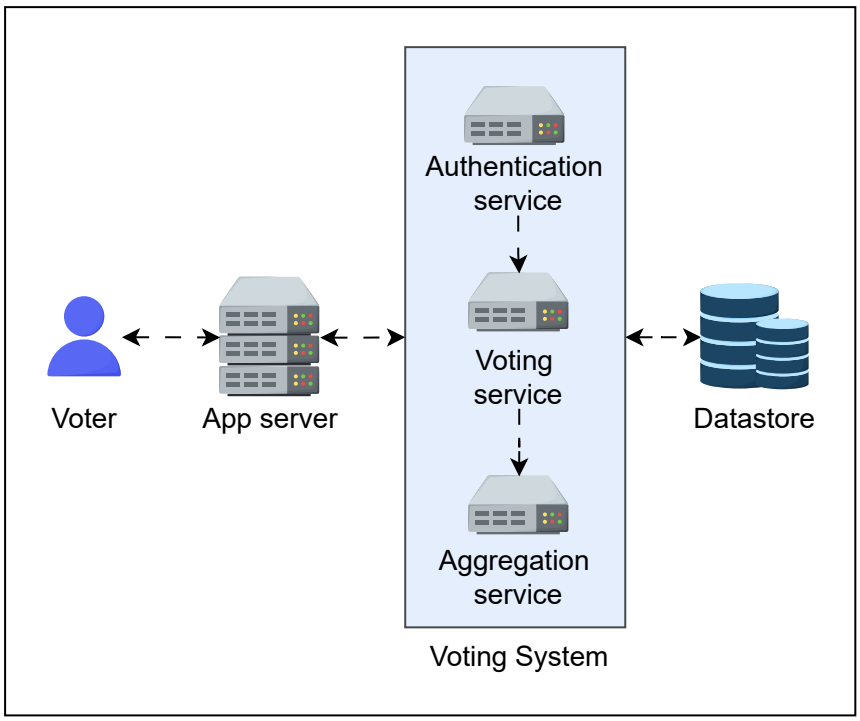
- **Functional requirements:**
  - **Voter authentication:** Verify voter identity through unique credentials.
  - **Vote:** Allow voters to cast votes securely.
  - **Vote storage:** Store encrypted votes in a secure, immutable database.
  - **Results:** Calculate and display results accurately after voting ends.
- **Nonfunctional requirements:**
  - **Anonymity:** Ensure voter's privacy by decoupling identities from votes.
  - **Transparency:** Ensure auditable and verifiable steps for registration, voting, and result publication while protecting voter privacy.
  - **Latency:** Ensure minimal delay during vote submission.

Now, let's look at how the system operates:

System workflow

In the electronic voting system, the voter logs into the system using authenticated credentials or biometric validation. After successful authentication, the voter casts their vote, encrypted and securely stored in an immutable database. The system prevents double voting and ensures

anonymity by separating voter identities from their votes. Once voting concludes, the encrypted votes are decrypted and aggregated in the aggregation service, and the results are published with an auditable trail for verification.



High-level design of electronic voting system

Test your knowledge!

- How does the electronic voting system prevent double voting while maintaining voter anonymity?

To learn how to tackle such questions during a System Design interview, check out the "Grokking the Modern System Design (https://www.educative.io/courses/grokking-the-system-design-interview)" course, which offers expert insights and examples.

21. Design an ATM system

Problem statement: Design an ATM (Automated Teller Machine) system that provides secure access to banking services, including cash withdrawal, balance inquiry, and fund transfers. The system must ensure robust security, seamless transaction execution, and a user-friendly interface for interacting with the ATM.

Test your knowledge!

- If a user withdraws money from an ATM but the machine runs out of cash, the amount is deducted from the user's account without dispensing the cash. How should the system handle this situation to maintain accurate transaction records?
- What measures should be implemented to detect and resolve such discrepancies in real time?

22. Design a multiplayer game

Problem statement: Design an online multiplayer game system that enables players to connect, interact, and compete in real-time across the globe. The system must ensure a seamless, engaging gaming experience by synchronizing player matchmaking and real-time gameplay.

Let's consider the following requirements for designing a multiplayer game:

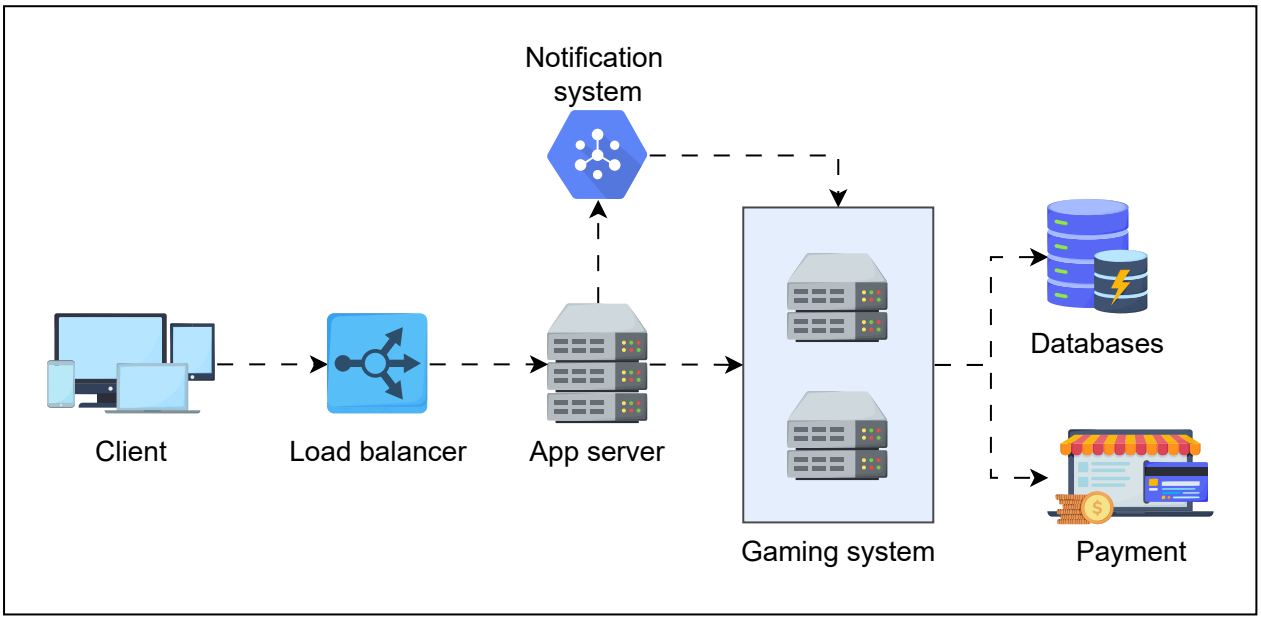
- **Functional requirements:**
  - **Gameplay (connect and play):** Allows players to connect with others and engage in real-time multiplayer gaming.
  - **Chat service:** Enable text-based communication between players during the gameplay.
  - **Event notification:** Send notifications to players about in-game events.
  - **Payment:** Facilitate secure transactions for purchasing in-game items.
- **Nonfunctional requirements:**
  - **Scalability:** Handle thousands of concurrent players.
  - **Latency:** Ensure minimal delay between players' actions and game state updates.
  - **Availability:** Highly available, with redundancy to prevent downtime during peak hours.

Avoid common pitfalls during System Design interviews. For insights, check out a mistake engineers often make in System Design interviews (https://www.educative.io/blog/mistakes-engineers-make-in-system-design-interviews) to strengthen your approach and stand out.

Let's see the system workflow of a multiplayer game:

System workflow

In the online multiplayer game system, once the user is logged in, the player selects a game mode, and the matchmaking service pairs them with players of similar skill levels. After a match is found, players enter a shared game lobby where they can prepare and communicate. Upon game start, player actions and game states are synchronized in real-time across all connected players, ensuring smooth gameplay. At the end of the match, the system updates leaderboards, saves the game results, and allows players to queue for another match.



A high-level design of an online multiplayer gaming system

Test your knowledge!

- How can matchmaking in a multiplayer game be handled to ensure balanced skill levels among players?
- What approaches would you take to prevent cheating and maintain fairness in the game?

To dive deeper into the details of a multiplayer game, explore the Gaming API (https://www.educative.io/courses/grokking-the-api-design-interview/requirements-of-the-gaming-api) design.

23. Design a parking lot system

Problem statement: Design a parking lot system that efficiently manages vehicle entry, parking, and exit. The system must support vehicle allocation, real-time availability tracking, and payment processing.

Test your knowledge!

- What are the functional and nonfunctional requirements for the parking lot?
- Should different vehicle types (e.g., cars, bikes, trucks) have separate entry and exit points?
- How will the system manage payment processing for various types (card, cash, etc.)?
- How would you visually represent specific activities, such as customers making payments for parking tickets or display panels indicating available spots?

24. Design an API system

Problem statement: Design an API system that serves as a communication layer between clients and backend services, enabling seamless interaction and data exchange for software applications. The API must be user-centric, ensuring scalability, security, and reliability to meet the demands of its users and applications.

Test your knowledge!

What are the functional and nonfunctional requirements for designing an API system?

Let's see the system workflow of an API:

System workflow

When a client sends a request, the API Gateway handles authentication and routes the request to the appropriate service. The service processes the request, interacts with the database if needed, and responds to the client.

Test your knowledge!

- What key resources will the API manage, and how should they be organized into endpoints?
- How will the API handle a high volume of requests and prevent abuse (e.g., rate limiting, throttling)?
- How will the API handle errors and ensure consistent, meaningful responses for developers?

To master answering any API design question with confidence, take the course "Grokking the Product Architecture Design Interview (https://www.educative.io/courses/grokking-the-api-design-interview)."

25. Design a real-time messaging system

Problem statement: Design a messaging system that allows users to send and receive text, images, videos, and other media types. The app must deliver messages quickly, ensure data security, and be scalable. In addition, real-time notifications, user authentication, and message synchronization across devices are essential for a seamless user experience.

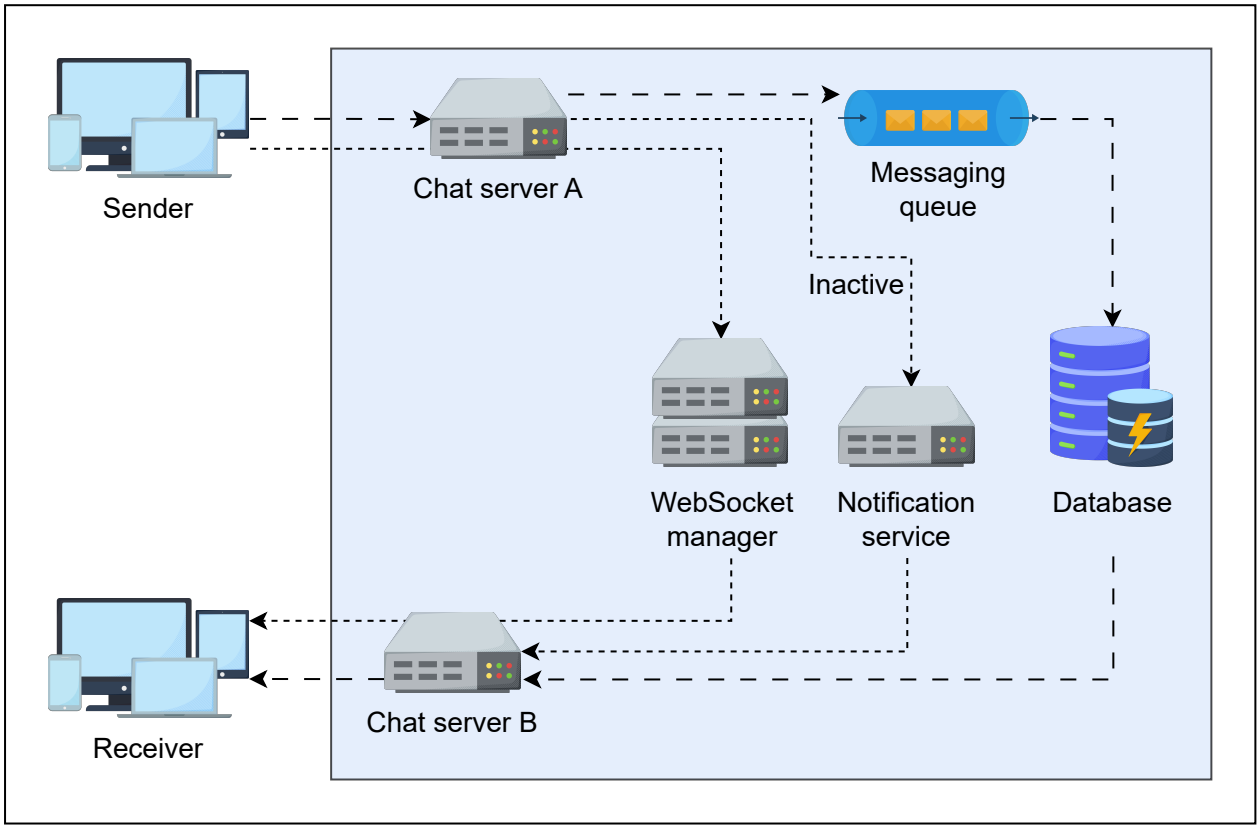
Let's consider the following requirements for designing a real-time messaging app:

- **Functional requirements:**
  - **User authentication:** Ensure secure login and account management for users.
  - **Real-time management:** Allow users to send and receive messages instantly with delivery and read receipts.
  - **Group chat support:** Enable users to create join, and interact in group conversations.
  - **File sharing:** Support sending and receiving multimedia files such as images, videos, and documents.
- **Nonfunctional requirements:**
  - **Scalability:** Handle a growing number of users and message traffic efficiently.
  - **Low latency:** Messages should be delivered with minimal delay.
  - **Reliability:** Ensure the system is highly available with fault-tolerant architecture.
  - **Security:** Implement end-to-end message encryption and secure data storage to protect user privacy.

Let's see the system workflow of a real-time messaging app:

System workflow

When a user sends a message, the app authenticates the user and routes the message to the server, which stores it in a database. If the recipient's app is active, the message is delivered instantly via WebSocket; if inactive, the server uses a notification service to alert the recipient. Once the recipient receives the notification or opens the app, the message is retrieved from the server and displayed in the conversation. Real-time updates, such as status changes, are pushed using web sockets.



A high-level design of messaging app

Test your knowledge!

- How will the senders and receivers connect with the servers and database to send and receive messages?
- How will you handle offline messages and sync them when the user is back online?

Explore the detailed design of a real-time communication system like WhatsApp (https://www.educative.io/courses/grokking-modern-system-design-interview-for-engineers-managers/system-design-whatsapp) on Educative.

26. Design an online book review system

Problem statement: Design a system that aggregates book reviews from external sources and integrates them into an online bookstore. The system should provide a diverse opinion on books to enhance the customer experience and assist users in making informed purchasing decisions.

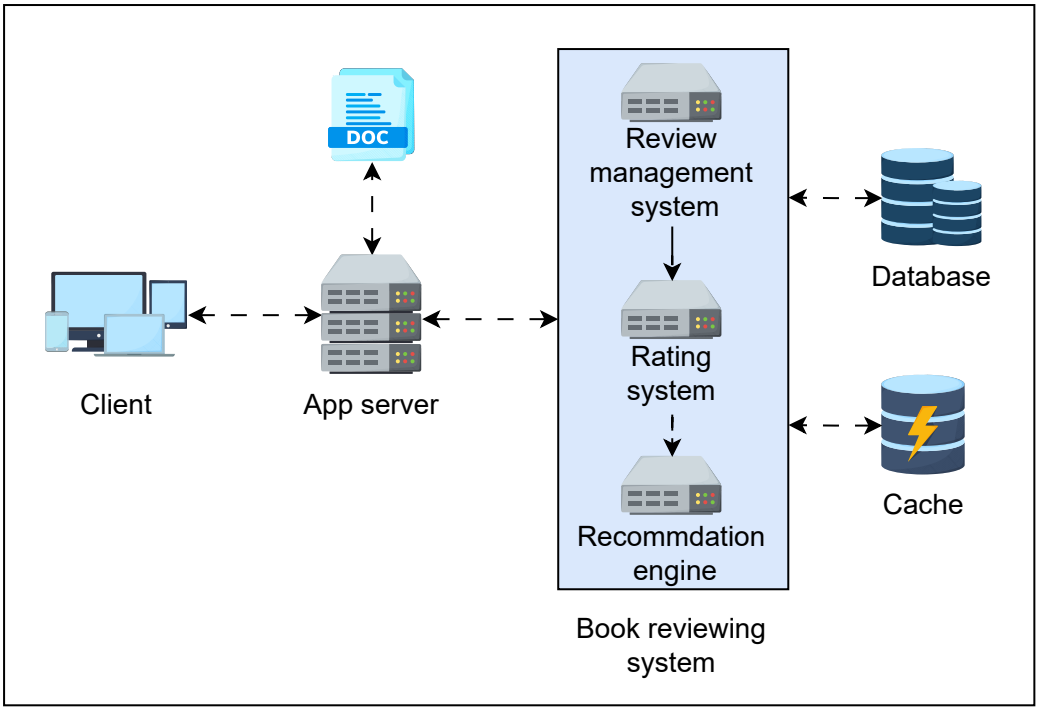
Let's consider the following requirements for designing an online book review system.

- **Functional requirements:**
    - **Reviews:** Ability to fetch reviews from external sources.
    - **Sorting:** Enable filtering and sorting reviews by relevance.
    - **Results:** Integrate reviews with the bookstore's product catalog.
    - **Updates:** Handle updates to reflect new reviews in real-time.
- **Nonfunctional requirements:**
    - **Scalability:** Manage reviews for an extensive catalog of books.
    - **Latency:** Ensure reviews are displayed quickly.
    - **Data integrity:** Prevent duplication or errors in displayed reviews.

The system workflow of online bookstore review is as follows:

System workflow

The online bookstore review system fetches external reviews periodically via API or web scraping. The fetched data is cleaned, deduplicated, and stored in a dedicated review database. The bookstore's frontend queries the review database to display relevant reviews for each book. Then, the review data is updated in real-time or on schedule to ensure the latest information is displayed to users.



High-level design of online bookstore review system

Test your knowledge!

- What approach would you take to handle rate-limiting issues when fetching reviews from external APIs?
- How would you ensure that the review data remains up-to-date?
- How can new reviews be aggregated from multiple sources without causing duplication or inconsistency when the number of books and reviews grows?

27. Design a content delivery network (CDN) system

Problem statement: Design a scalable content delivery network system that efficiently distributes and caches content across globally distributed servers. The system must minimize latency and ensure reliable content delivery to end users, regardless of their geographic location.

Let's consider the following requirements for designing a CDN system:

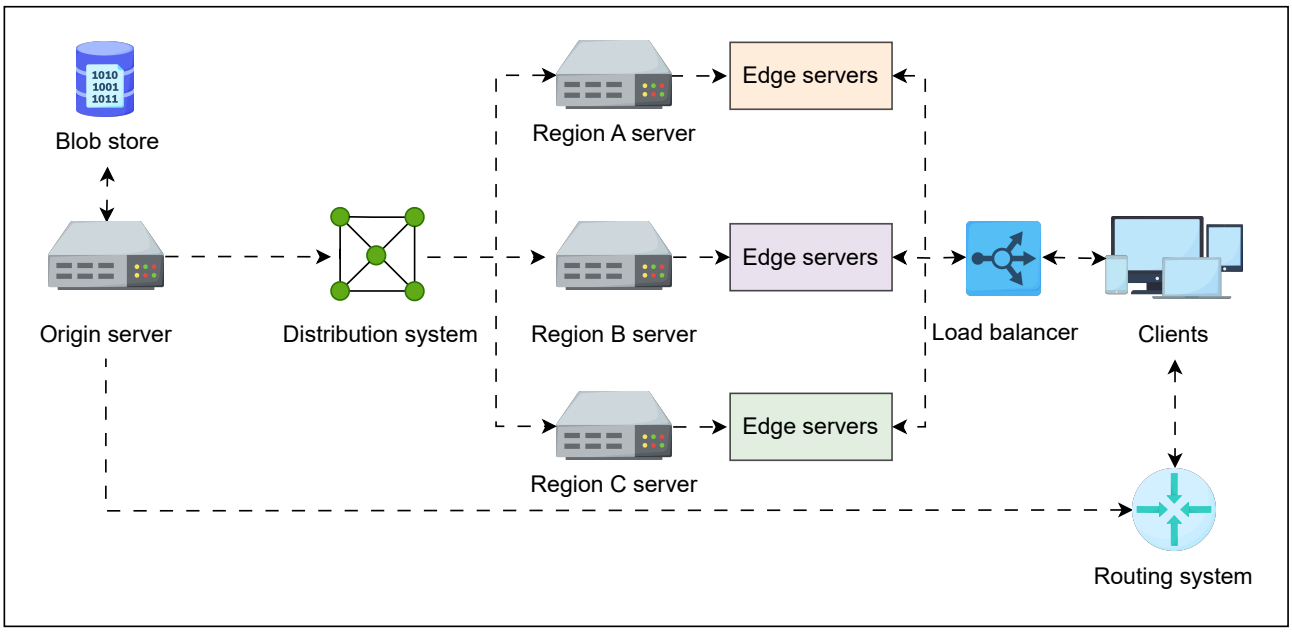
- **Functional requirements:**
  - **Content retrieval:** Retrieve content from the origin server.
  - **User request handling:** Respond to user requests with minimal latency.

- **Auto content delivery:** Automatically deliver cached content from edge servers to users.
- **Content update:** Update content from the origin or peer CDN to keep content updated.

- **Nonfunctional requirements:**
  - **Scalability:** Handle growing traffic by adding more servers.
  - **Reliability:** Consistently deliver content without failures or delays.
  - **Security:** Protect user data and content using encryption and access controls.

System workflow

In the CDN system, when a client requests content, the request routing system determines the nearest or fastest server to minimize latency. A load balancer directs the request to this server, which checks if the content is cached. If the content is in the cache, it is served immediately; if not, the server retrieves it from the origin, stores it in the cache, and then delivers it to the client. The CDN periodically updates and purges less popular content while continuously monitoring performance for optimization.



A high-level design of a CDN

Test your knowledge!

- How can the CDN infrastructure be guaranteed scalability, reliability, and fault tolerance?

To answer more such interesting questions, visit the detailed design of the content delivery network (https://www.educative.io/courses/grokking-the-system-design-interview/system-design-the-content-delivery-network-cdn).

28. Design a forum-like system

Problem statement: Design a forum-like system (like Quora, Reddit, or HackerNews) where users can post questions and answers, vote, comment, and search for content. The system should include features for ranking answers based on quality and relevance and personalized content for each user.

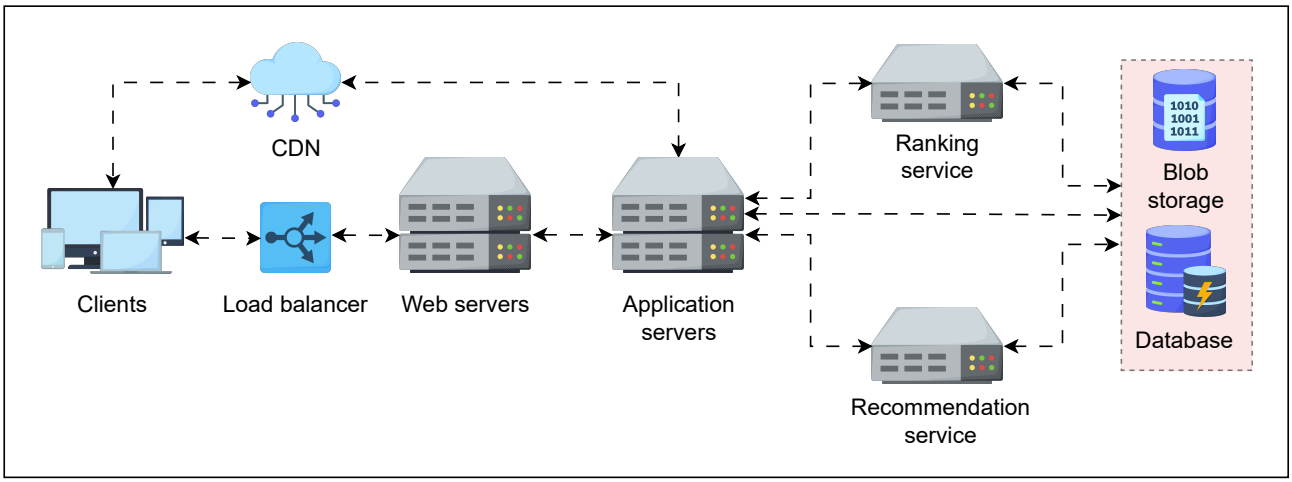
Let's consider the following requirements for designing a forum-like system:

- **Functional requirements:**
  - **Post questions and answers:** Allow users to create and respond to questions.
  - **Votes:** Enable users to upvote, downvote, and comment on questions and answers.
  - **Search:** Provide a search functionality to find questions, answers, and users.
  - **Recommendation system:** Suggest personalized content based on user interests and behavior.
- **Nonfunctional requirements:**
  - **Scalability:** Handle increasing users and content while maintaining data.
  - **Availability:** Users can access content at all times.
  - **Performance:** Interactions like posting, voting, and fetching content should be in minimal time.

Now, let's see the system workflow of the forum-like system:

System workflow

In a forum-like system, users interact through a web server, which communicates with an application server to manage actions like posting questions, answers, and comments. Media content is stored in blob storage, while structured data such as Q/A and user profiles are stored in a relational database like MySQL. A machine learning engine evaluates user actions to rank answers, continuously learning from feedback to improve accuracy. A recommendation service tailors content based on user preferences, enhancing their experience by suggesting relevant questions and answers.



A high-level design of forum-like system

> **Note:** To understand the forum-like system, refer to the Quora System Design (https://www.educative.io/courses/grokking-the-system-design-interview/system-design-quora).

29. Design a web crawler system

Problem statement: Design a web crawler system that automatically browses and indexes web pages for search engine indexing, data analysis, or website monitoring. The system must efficiently gather and process web data while adhering to crawl restrictions, ensuring scalability, and handling diverse website structures.

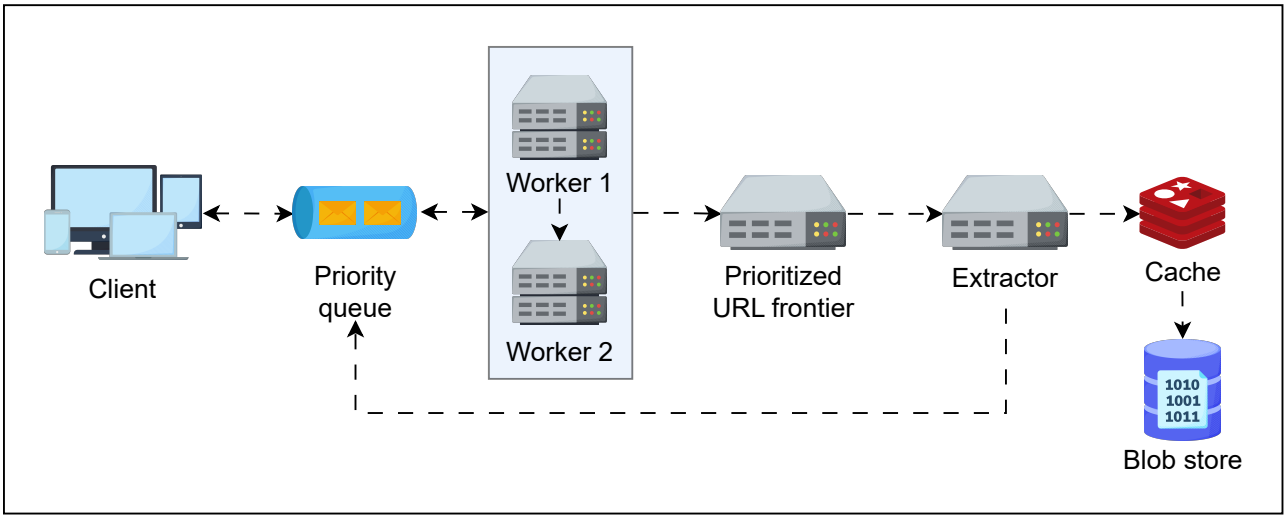
Let's consider the following requirements for designing a web crawler:

- **Functional requirements:**
  - **Crawl web pages:** Fetch web pages starting from a seed URL and follow hyperlinks to discover new pages.
  - **Content parsing:** Extract relevant information from web pages.
  - **URL prioritization:** Decide the order in which pages are crawled.
  - **Data storage:** Store the crawled data in an SQL database for easy retrieval
- **Nonfunctional requirements:**
  - **Performance:** Optimize the speed of crawling and processing.
  - **Fault tolerance:** The system should recover from errors without losing data.
  - **Efficiency:** Minimize bandwidth usage and avoid unnecessary downfalls.

Let's see how the web crawler works:

System workflow

The web crawler starts by fetching a seed URL and parsing the page to extract hyperlinks and metadata. These new URLs are added to a prioritized URL Frontier, managed by the crawler schedule service, which assigns priorities based on domain reputation and link depth. The crawler checks `robots.txt` for permissions and uses duplicate detection to avoid re-crawling the same pages. Crawled data is stored in a database for further use, while the system ensures scalability, fault tolerance, and adherence to web standards.



A high-level design of a web crawler

Test your knowledge!

- In real-world web crawlers, multiple workers process different URLs concurrently. How does this impact the design and management of the queuing system?

To get answers to more such interesting questions, refer to the web crawler System Design (https://www.educative.io/courses/grokking-the-system-design-interview/system-design-web-crawler).

30. Design a ZooKeeper service

Problem statement: Design a distributed coordination service like ZooKeeper to manage configuration, synchronization, naming, and distributed locks across a cluster. The system should provide a hierarchical data model for storing small amounts of configuration data.

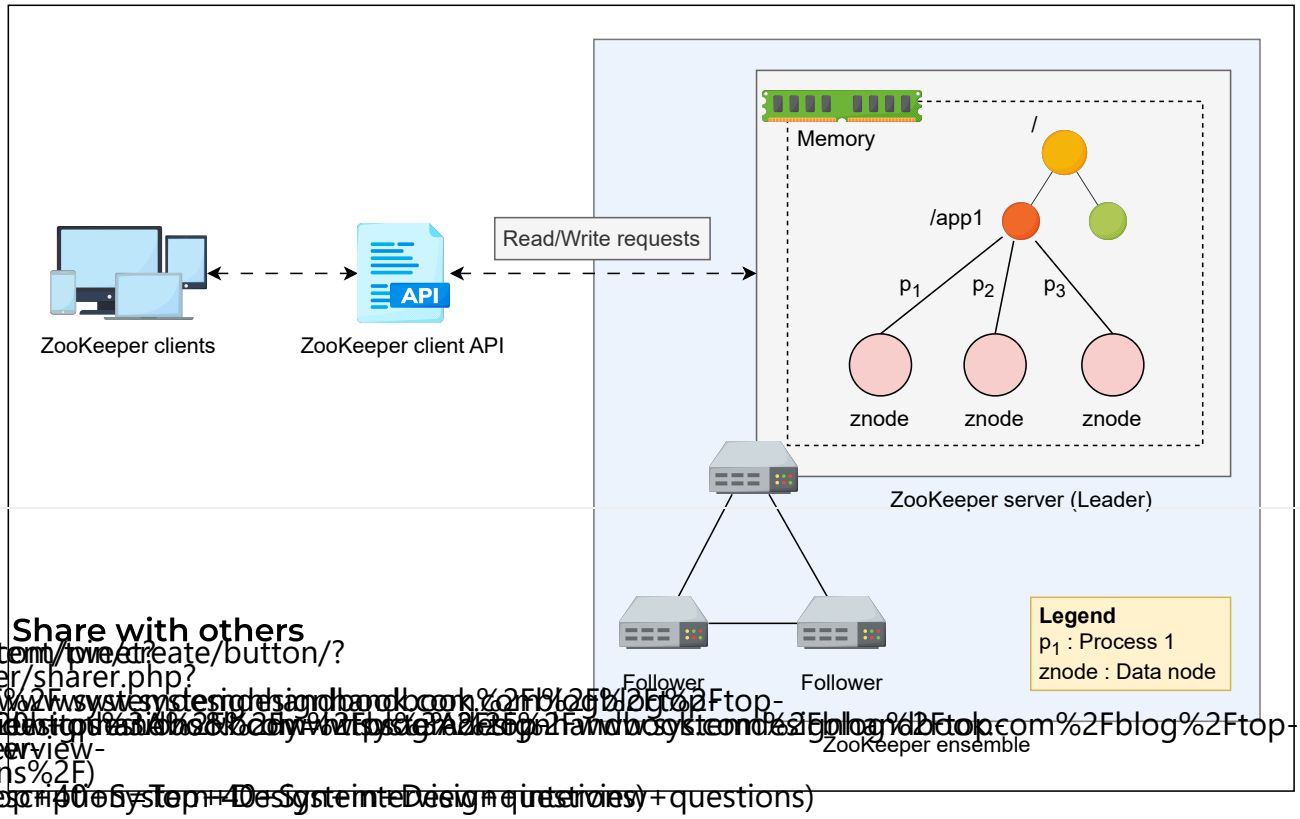
Let's consider the following requirements for designing a zooKeeper:

- **Functional requirements:**
    - **Configuration management:** Store and manage configuration data for distributed systems.
    - **Naming service:** Maintain and resolve unique node names in a distributed environment.
    - **Leader election:** Facilitates leader election for distributed systems.
    - **Data watch:** Notify clients of changes in data stored in ZooKeeper.
- **Nonfunctional requirements:**
    - **Consistency:** Ensure strong consistency across the system.
    - **Availability:** Provide high availability with minimal downtime.
    - **Reliability:** Ensure data persistence and fault tolerance.
    - **Scalability:** Handle increasing client loads and large clusters.

Now, let's look at how the system operates:

System workflow

The ZooKeeper client API interacts with the system through the ZooKeeper client API, which provides operations like create, delete, and exist to manage coordination data stored in z*nodes*. Client requests are forwarded to a ZooKeeper server, which organizes the data hierarchically (like a file system) and stores it in memory. A ZooKeeper server ensemble consists of multiple replicated servers, with one server as the leader and others as followers.



A high-level design of a ZooKeeper system

April 16, 2025 Fahim Ul Haq 41 min read

Note: Explore more for a detailed design of ZooKeeper (https://www.educative.io/courses/grokking-the-principles-and-practices-of-advanced-system-design/introduction-to-zookeeper).

Leave a Reply

Additional questions

Your email address will not be published. Required fields are marked *

Here are some additional questions you can practice or keep in mind to sharpen your skills:

Comment *

31. Design a distributed storage system like Bigtable (https://www.educative.io/courses/grokking-the-principles-and-practices-of-advanced-system-design/introduction-to-bigtable)

32. Design an API rate limiter for sites like Firebase (https://www.educative.io/courses/grokking-the-system-design-interview/system-design-the-rate-limiter)

33. Design a DoorDash system (https://www.educative.io/blog/microsoft-system-design-interview#Design-a-food-delivery-systemUber-Eats)

34. Design distributed locking service. (https://www.educative.io/courses/grokking-the-principles-and-practices-of-advanced-system-design/introduction-to-chubby)

Name *

35. Design an authentication system (https://www.educative.io/courses/grokking-the-api-design-interview/authentication-and-authorization)

36. Design location-based services like Yelp (https://www.educative.io/courses/grokking-the-system-design-interview/system-design-yelp)

Email *

37. Design notification service (https://www.educative.io/courses/grokking-the-system-design-interview/system-design-the-pub-sub-abstraction)

Website

38. Design a system to interview candidates (https://www.educative.io/courses/grokking-the-api-design-interview/requirements-of-the-leetcode-api)

39. Design code deployment system

Post Comment

40. Design a stock exchange syste

It's not the end! Continue exploring more design problems to prepare for your System Design interview.

Related Blogs

Wrap up

Mastering these 40 System Designs is not about clearing interviews but building the mindset and skills to tackle real-world challenges creatively. For those eager to dive deeper, explore:

Grokking the System Design Interview Course (https://www.systemdesignhandbook.com/blog/grokking-the-system-design-interview/)

- Grokking the Modern System Design Interview (https://www.educative.io/courses/grokking-the-system-design-interview)
- Grokking the Frontend System Design Interview (https://www.educative.io/courses/grokking-frontend-system-design-interview)
- Grokking the Generative AI System Design (https://www.educative.io/courses/generative-ai-system-design)
- Grokking the Product Architecture Design Interview (https://www.educative.io/courses/grokking-the-api-design-interview)
- System Design Deep Dive: Real-World Distributed Systems (https://www.educative.io/courses/system-design-deep-dive-real-world-distributed-systems)

System Design interviews aren't just tough—they're ambiguous by design. To succeed, you need more than textbook answers. You need to think like an architect under pressure. Search for "Grokking the

READ THE BLOG (HTTPS://WWW.SYSTEMDESIGNHANDBOOK.COM/BLOG/GROKKING-THE-SYSTEM-DESIGN-INTERVIEW/)

Best System Design Interview Prep (https://www.systemdesignhandbook.com/blog/best-system-design-interview-prep/)

These courses offer an in-depth understanding of core principles, practical examples, and detailed walkthroughs of various System Design problems. With these resources, you'll be well-equipped to confidently approach any System Design Interview and craft solutions that stand out.

If you're gearing up for a system design interview, chances are you've already asked yourself the big question: What is the best system design interview prep strategy? System design interviews

With dedication, practice, and the right tools, you're on the path to acing your System Design interviews and excelling in your career.

Best of luck!

READ THE BLOG (HTTPS://WWW.SYSTEMDESIGNHANDBOOK.COM/BLOG/BEST-SYSTEM-DESIGN-INTERVIEW-PREP/)

Take **mock interviews** (https://www.educative.io/mock-interview) to enhance your skills and prepare for real-world challenges. It's an excellent way to scale your knowledge, identify gaps, and build confidence for System Design interviews. Don't miss the opportunity to level up!

10 Best System Design Courses to Crack the Interview in 2025 (https://www.systemdesignhandbook.com/blog/best-system-design-courses/)

System design is one of the most critical skill sets for software engineers aiming to grow into senior roles. Whether you're preparing for high-stakes interviews at companies like Google or

READ THE BLOG (HTTPS://WWW.SYSTEMDESIGNHANDBOOK.COM/BLOG/BEST-SYSTEM-DESIGN-COURSES/)

(https://www.systemdesignhandbook.com)

QUICK LINKS

System Design Interview (https://www.systemdesignhandbook.com/guides/system-design-interview/)

System Design Interview Handbook (https://www.systemdesignhandbook.com/system-design-interview-handbook/)

System Design Interview Questions (https://www.systemdesignhandbook.com/blog/system-design-interview-questions/)

LEGAL

Privacy Policy (https://www.systemdesignhandbook.com/privacy/)

Terms of Service (https://www.systemdesignhandbook.com/terms/)