High-Level Design: Understanding how to design scalable, maintainable systems.

Low-Level Design: Creating detailed class diagrams, interaction diagrams, and defining APIs.

Key Concepts: Load balancing, caching, sharding, CAP theorem, microservices architecture, and database schema design.

System design interview questions are aimed at assessing your ability to design large-scale, scalable, reliable, and maintainable software systems. Here are some famous and commonly asked system design interview questions:

**1. Design a URL Shortener (e.g., TinyURL or Bitly)**

* Requirements: Shorten a URL, expand a shortened URL, handle high traffic, and provide analytics.
* Challenges: Data storage, collision avoidance, scalability, and expiration of unused URLs.

**2. Design an Online Bookstore (e.g., Amazon)**

* Requirements: Search for books, add to cart, manage inventory, recommendations, and user reviews.
* Challenges: Database schema, catalog management, user sessions, and recommendation engines.

**3. Design a Chat System (e.g., WhatsApp or Slack)**

* Requirements: One-on-one messaging, group chats, real-time delivery, notifications, and message history.
* Challenges: Message queuing, synchronization, end-to-end encryption, and scalability.

**4. Design a Social Media Platform (e.g., Twitter or Instagram)**

* Requirements: User timelines, follow/unfollow functionality, posting content, likes/comments, and trending topics.
* Challenges: Feed generation, data storage, consistency, and caching.

**5. Design a Ride-Sharing Service (e.g., Uber or Lyft)**

* Requirements: Driver-rider matching, pricing, real-time location updates, and ride status tracking.
* Challenges: Geolocation handling, dynamic pricing, load balancing, and fraud detection.

**6. Design a Video Streaming Service (e.g., YouTube or Netflix)**

* Requirements: Upload and stream videos, search functionality, recommendations, and user profiles.
* Challenges: Video encoding, CDN for streaming, personalization, and large-scale data handling.

**7. Design an E-Commerce Website**

* Requirements: User authentication, product listing, cart, checkout, payment, and order management.
* Challenges: Scalability, security (payment handling), and inventory updates.

**8. Design a Search Engine (e.g., Google)**

* Requirements: Crawl, index, and rank web pages, handle queries, and return results quickly.
* Challenges: Web crawling, indexing, ranking algorithms, and handling a large amount of data.

**9. Design a File Storage Service (e.g., Dropbox or Google Drive)**

* Requirements: File upload/download, sharing, versioning, and synchronization.
* Challenges: Data replication, consistency, handling large files, and offline support.

**10. Design a Notification System (e.g., for Email, SMS, or Push)**

* Requirements: Send notifications, prioritize messages, handle retries, and ensure delivery.
* Challenges: Reliability, scalability, and ensuring low latency.

**11. Design a Payment Gateway (e.g., Stripe or PayPal)**

* Requirements: Handle transactions, support multiple payment methods, and ensure security.
* Challenges: Fraud detection, scalability, and integration with banks and vendors.

**12. Design a Rate Limiter**

* Requirements: Control the number of requests per user or IP address within a time window.
* Challenges: Real-time enforcement, distributed systems, and scalability.

**13. Design a News Feed System (e.g., Facebook News Feed)**

* Requirements: Aggregate and rank content from friends and followed accounts.
* Challenges: Ranking algorithms, personalization, and caching strategies.

**14. Design a Distributed Caching System (e.g., Memcached or Redis)**

* Requirements: Improve application speed by caching frequently accessed data.
* Challenges: Cache eviction policies, consistency, and handling cache misses.

**15. Design a Calendar System (e.g., Google Calendar)**

* Requirements: Create, update, and share events, and manage notifications.
* Challenges: Synchronization, conflict resolution, and multi-time zone support.

**16. Design an API Rate Limiter**

* Requirements: Throttle API requests per user/IP address to prevent abuse.
* Challenges: Token bucket algorithms, distributed enforcement, and scalability.

**17. Design a Ticket Booking System (e.g., Movie Tickets or Airline Reservations)**

* Requirements: Search for tickets, seat selection, payment, and manage reservations.
* Challenges: Concurrency, overbooking, and handling peak traffic.

**18. Design a Monitoring System (e.g., Prometheus or DataDog)**

* Requirements: Collect, store, and visualize system metrics; alert on anomalies.
* Challenges: Data ingestion, real-time analytics, and scalability.

**19. Design a Multi-player Online Game System**

* Requirements: Real-time communication, synchronization, matchmaking, and leaderboards.
* Challenges: Latency, consistency, and fault tolerance.

**20. Design a Hotel Booking System**

* Requirements: Search rooms, check availability, make reservations, and handle cancellations.
* Challenges: Database design, real-time availability updates, and concurrency control.

For each of these questions, practice by identifying:

1. Functional and non-functional requirements.
2. High-level architecture (clients, servers, databases, etc.).
3. Components (APIs, load balancers, cache, etc.).
4. Trade-offs (scalability, consistency, latency, etc.).
5. Challenges and solutions.

Would you like to discuss or design any specific one?

In the FinTech domain, system design questions often revolve around payment systems, fraud detection, trading platforms, financial data analysis, and scalability of critical systems. Here are some domain-specific examples:

**1. Design a Payment Gateway (e.g., Stripe or PayPal)**

* **Requirements:**
  + Process payments using various methods (credit card, bank transfer, wallets).
  + Support currency conversion.
  + Ensure transaction security (PCI DSS compliance).
  + Handle refunds and chargebacks.
* **Challenges:**
  + High availability, scalability during peak loads, and fraud detection.
  + Integration with banks and compliance with regulatory requirements.

**2. Design a Stock Trading Platform (e.g., Robinhood or Zerodha)**

* **Requirements:**
  + Real-time stock price updates and market data visualization.
  + Buy/sell functionality with order types (market, limit, stop).
  + Support for margin trading, options, and derivatives.
* **Challenges:**
  + Low-latency order matching, data consistency, and risk management.
  + Handling high transaction volumes during market volatility.

**3. Design a Fraud Detection System**

* **Requirements:**
  + Analyze transaction patterns to detect fraudulent activities.
  + Trigger alerts or block suspicious transactions.
  + Real-time fraud scoring and machine learning integration.
* **Challenges:**
  + Balancing false positives and false negatives.
  + Scalability to handle high transaction volumes.
  + Ensuring minimal latency in real-time systems.

**4. Design a Loan Management System**

* **Requirements:**
  + Manage customer applications, credit scoring, and loan disbursements.
  + Track EMI payments and provide repayment schedules.
  + Support multiple loan types (personal, auto, home).
* **Challenges:**
  + Integration with credit bureaus and real-time risk assessment.
  + Scalability to handle large customer bases and financial computations.

**5. Design a Personal Finance Management App (e.g., Mint)**

* **Requirements:**
  + Aggregate user accounts across banks and track expenses.
  + Categorize transactions, provide budgeting tools, and offer insights.
  + Display financial trends and suggest investment opportunities.
* **Challenges:**
  + Securely connecting to third-party financial institutions.
  + Handling sensitive data with encryption and regulatory compliance.
  + Scalability for a growing user base.

**6. Design a Cryptocurrency Exchange Platform**

* **Requirements:**
  + Support buying, selling, and trading cryptocurrencies.
  + Provide wallets for secure crypto storage.
  + Enable real-time market data and charting tools.
* **Challenges:**
  + Security (cold storage, DDoS protection), compliance with regulations.
  + Handling volatility and scalability for high trading volumes.

**7. Design a Peer-to-Peer Lending Platform**

* **Requirements:**
  + Match borrowers with lenders based on risk profiles.
  + Automate loan agreements and repayment schedules.
  + Provide insights into investment risk and expected returns.
* **Challenges:**
  + Risk assessment, fraud detection, and regulatory compliance.
  + Scalability for large numbers of borrowers and lenders.

**8. Design a Financial Data Aggregator (e.g., Plaid)**

* **Requirements:**
  + Aggregate data from multiple banks, credit card providers, and brokerages.
  + Provide APIs to clients for accessing financial data.
  + Ensure real-time synchronization.
* **Challenges:**
  + Secure data transmission, maintaining data integrity, and compliance with regulations.
  + Handling rate-limited APIs and ensuring uptime.

**9. Design a Digital Wallet System (e.g., Google Pay, Paytm)**

* **Requirements:**
  + Store funds securely, support QR code payments, and handle transfers.
  + Provide loyalty rewards and cashback programs.
  + Ensure interoperability with banks and financial institutions.
* **Challenges:**
  + Handling large-scale payment processing and regulatory compliance.
  + Preventing misuse and ensuring secure authentication.

**10. Design a BNPL (Buy Now Pay Later) System**

* **Requirements:**
  + Integrate with e-commerce platforms for deferred payment options.
  + Assess customer creditworthiness in real time.
  + Handle installment payments and manage defaulters.
* **Challenges:**
  + Risk analysis and underwriting for instant credit approval.
  + Scalability during shopping festivals or sales events.

**11. Design a Cross-Border Payment System**

* **Requirements:**
  + Facilitate international money transfers with currency conversion.
  + Ensure compliance with anti-money laundering (AML) and KYC regulations.
  + Minimize transaction fees and transfer delays.
* **Challenges:**
  + Handling different regulatory requirements in multiple countries.
  + Currency volatility and integration with international banks.

**12. Design a Credit Scoring System**

* **Requirements:**
  + Aggregate data from multiple sources (banks, utilities, credit bureaus).
  + Calculate a risk score for loan approvals or credit cards.
  + Provide dynamic scoring based on user behavior changes.
* **Challenges:**
  + Integrating diverse data sources and ensuring score accuracy.
  + Maintaining data privacy and compliance.

Would you like to explore any of these in depth?

For a **UI Architect** in the FinTech domain, system design questions often focus on architecting scalable, secure, and user-friendly interfaces while considering accessibility, performance, and integration with backend systems. Here are some design problems tailored for a UI Architect:

**1. Design a Dashboard for a FinTech Platform**

* **Scenario:** A dashboard that displays a user's financial summary, such as account balances, recent transactions, investment performance, and alerts.
* **Focus Areas:**
  + Responsive and adaptive UI for different devices.
  + Real-time data updates using WebSockets or polling.
  + Modular design for widgets (e.g., customizable and draggable components).
  + Accessibility (ARIA roles, keyboard navigation).
  + Handling large datasets efficiently with virtual scrolling or pagination.
* **Challenges:**
  + Maintaining a smooth user experience under high data load.
  + Ensuring security for sensitive financial information.

**2. Design a Multi-Step Loan Application Flow**

* **Scenario:** Build a wizard-like interface for applying for loans with steps for personal information, income details, credit checks, and submission.
* **Focus Areas:**
  + State management across multiple steps.
  + Real-time validation of form inputs.
  + Integration with APIs for credit scoring and KYC.
  + Saving and resuming incomplete applications.
* **Challenges:**
  + Designing for error handling and edge cases (e.g., failed API calls).
  + Supporting internationalization and localization.

**3. Architect a Notification System UI**

* **Scenario:** Display notifications for transactions, account updates, and promotional offers in real time.
* **Focus Areas:**
  + A unified notification panel with filters and categories (e.g., unread, urgent).
  + Real-time updates using a push service or WebSockets.
  + Accessibility for screen readers and keyboard shortcuts.
  + Configurability for users to customize notification preferences.
* **Challenges:**
  + Optimizing the UI for a large volume of notifications.
  + Ensuring consistency across web and mobile interfaces.

**4. Design a Trading Platform Interface**

* **Scenario:** A real-time trading platform showing stock prices, buy/sell options, and portfolio performance.
* **Focus Areas:**
  + High-frequency updates for market data with minimal latency.
  + Interactive charts with zoom, pan, and tooltips.
  + User-friendly design for placing orders quickly.
  + Dark mode for professional users.
* **Challenges:**
  + Rendering high-performance data visualizations.
  + Handling errors or delays in real-time data updates.

**5. Architect a Modular Design System for FinTech Applications**

* **Scenario:** Develop a reusable design system for building consistent UIs across multiple FinTech products.
* **Focus Areas:**
  + Component library with standardized styling (buttons, inputs, cards).
  + Accessibility compliance (color contrast, focus management).
  + Support for theming (light/dark mode, brand customization).
  + Integration with front-end frameworks like Angular or React.
* **Challenges:**
  + Ensuring scalability as the design system grows.
  + Supporting multiple brands or themes within the same product ecosystem.

**6. Design a Secure Authentication Flow UI**

* **Scenario:** Build a secure and user-friendly login flow for a FinTech application.
* **Focus Areas:**
  + Support for multi-factor authentication (e.g., OTP, biometrics).
  + Progressive disclosure of additional security steps.
  + Handling edge cases like forgotten passwords and account locks.
  + Providing feedback for incorrect credentials.
* **Challenges:**
  + Balancing security and usability.
  + Designing fallback flows for users without access to MFA devices.

**7. Design a Reporting and Analytics Dashboard**

* **Scenario:** Provide a detailed analytics interface for financial advisors or businesses to track transactions and performance.
* **Focus Areas:**
  + Interactive charts and tables for detailed analysis.
  + Filters for date ranges, account types, and regions.
  + Downloadable reports in multiple formats (PDF, Excel).
  + Configurable widgets for personalization.
* **Challenges:**
  + Ensuring smooth performance with complex queries and large datasets.
  + Supporting accessibility and mobile-first design.

**8. Architect a Low-Code UI for Financial Configurations**

* **Scenario:** Enable business users to create and manage financial rules, workflows, or configurations without developer intervention.
* **Focus Areas:**
  + Drag-and-drop interfaces for building workflows or forms.
  + Validation to prevent incorrect configurations.
  + Integration with backend services for execution.
  + Role-based access control to protect sensitive features.
* **Challenges:**
  + Balancing simplicity for non-technical users with advanced capabilities.
  + Ensuring security and auditability of configurations.

**9. Design a Customer Support Chat Interface**

* **Scenario:** Build a chat interface for resolving customer queries about transactions, account issues, or loans.
* **Focus Areas:**
  + Real-time messaging with typing indicators and chat history.
  + Integration with AI chatbots for FAQs and human agents for escalations.
  + Feedback collection after chat sessions.
  + Multi-language support.
* **Challenges:**
  + Ensuring a seamless handoff between chatbots and agents.
  + Designing for mobile-first experiences.

**10. Architect a Multi-Currency Conversion Tool UI**

* **Scenario:** Provide an interface for users to view and convert currency rates.
* **Focus Areas:**
  + Interactive currency selectors with real-time rate updates.
  + Displaying transaction fees and final amounts clearly.
  + Support for both simple and advanced use cases (e.g., batch conversions).
* **Challenges:**
  + Handling real-time updates without performance issues.
  + Designing for localization (currency formats, languages).

Would you like to deep-dive into any of these examples or discuss how they might align with your responsibilities?