Mastering System Design

Design an Auction Platform (aka eBay)

What is an Auction Platform?

- A digital system that enables users to list items and bid in real-time within a defined time window.
- Auctions promote competitive pricing and urgency — the highest bid at closing time wins.
- The platform handles item listings, bid tracking, real-time updates, auction lifecycle, and payment processing.
- Key Objective:
    a. Deliver a secure, fair, and scalable real-time bidding experience — from listing to payment.

Functional Requirements

- User registration & authentication
- Item listing with auction parameters (start/end time, reserve price)
- Real-time bid placement and updates
- Auction state transitions: scheduled → active → ended
- Payment processing after auction ends (e.g., via Stripe, PayPal)
- Notifications: outbid, auction won, payment pending/complete

Key Actors & Use Cases

- 👤 Actors:
  - Seller: Lists items, sets pricing rules
  - Bidder: Places bids, gets real-time notifications
  - System: Handles auction logic, enforces timing, manages payments
  - Admin: Monitors platform activity and handles fraud or disputes
- 📌 Use Cases:
  - Seller lists a gaming console for a 3-day auction
  - Multiple bidders join and bid competitively
  - System tracks bids, updates UI in real-time
  - Winning bidder completes payment → seller is notified

Non-Functional Requirements

- ⚙️ Performance:
  - Sub-second latency for bid placement and bid updates
- 📈 Scalability:
  - Support thousands of concurrent auctions and real-time users
- 🔒 Security:
  - Secure user authentication, payment data protection, anti-bot protections
- 🕑 Availability:
  - Ensure high availability during peak auction times and payment flow
- 📊 Observability:
  - Logging of auction and payment events, real-time monitoring dashboards

Constraints & Challenges

- ⏱️ Real-Time Pressure:
  - Handle bids submitted within milliseconds of closing — resolve ties & order events accurately
- ⚔️ Concurrency Conflicts:
  - Simultaneous bid updates must be conflict-free and idempotent
- 📡 Live Updates:
  - Efficient, scalable real-time delivery to all watchers of an auction (via WebSockets/pub-sub)
- 💸 Payments Workflow:
  - Payment failures, timeouts, retries, and fraud detection must be handled gracefully
- 🧠 Fairness & Trust:
  - Prevent bots/sniping and ensure fair auction rules are enforced transparently

Estimating Scale

- Assumed Metrics:
  - 👫 Users: 5M registered users, 500K DAU
  - 📦 Active Listings: 1M ongoing auctions at any moment
  - 💬 Bids: Avg. 10 bids/auction → 10M bids/day
  - ⏱️ Peak Activity: 10K concurrent users bidding/viewing one popular auction
  - 💸 Payments: 100K completed auctions/day → 100K payment transactions/day

Traffic Patterns & Real-Time Pressure Points

- 🔁 Read-Heavy Operations:
  - Viewing auction listings & item details
  - Real-time bid updates to watchers
  - Browsing user profiles, search, filters
  - Scale: Millions of reads per minute at peak
- ✍️ Write-Sensitive Operations:
  - Bid placement (needs low latency, strong consistency)
  - Creating new auction listings
  - Finalizing auctions & triggering payments
  - Write volume is lower but time-critical — especially near auction end
- ⚡ Real-Time Pressure Zones:
  - Last-minute bidding: thousands of bids in final seconds
  - Real-time fan-out: pushing updates to thousands of watchers
  - Closing auctions precisely (scheduler must be accurate)
  - Payment triggers: time-sensitive + 3rd-party API dependent

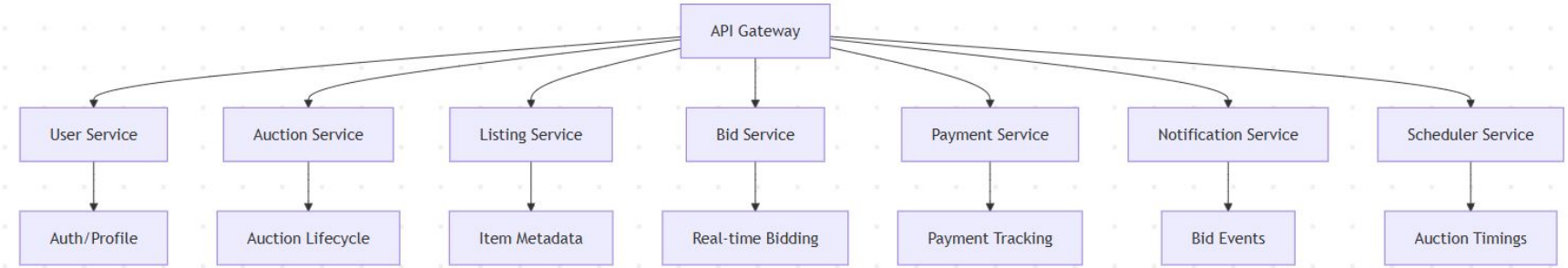
Takeaway: The system must scale for reads, but remain fast and correct for writes under pressure.

Identifying System Bottlenecks and Challenges

- Estimate per-service load to prevent surprise bottlenecks
- Isolate real-time components for better scalability
- Use async processing where consistency can be relaxed (e.g., email, payment receipt generation)
- Use horizontal scaling + partitioning for high-volume data (e.g., bids, listings)
- Plan for “hot” auctions — design for skewed load

High-Level Architecture Overview and Key components

- 🌐 API Gateway – Entry point, request routing, rate limiting
- 👥 User Service – Auth, profile, roles (buyer/seller)
- 📦 Auction Service – Manages auction lifecycle, bid validation
- 🏷️ Listing Service – Handles item metadata, media, categories
- ⚡ Bid Service – Real-time bid management, concurrency control
- 🧾 Payment Service – Triggers post-auction payments, payment status tracking
- 📣 Notification Service – Sends bid events, outbid alerts, auction results
- 🧭 Scheduler Service – Handles auction start/end timings
- 📊 Analytics & Logging – Tracks usage, bids, auction trends



API Design – Key Endpoints

- User APIs:
  - POST /signup, POST /login
  - GET /user/profile
- Auction APIs:
  - POST /auctions – Create new auction
  - GET /auctions/{id} – View auction
  - POST /auctions/{id}/bids – Place bid
  - GET /auctions/{id}/bids – Bid history
  - GET /auctions/active – List active auctions
- Payment APIs:
  - POST /payments/initiate – Trigger post-auction
  - GET /payments/{id}/status
- Notification Triggers (Internal):
  - Auction ending → notify winner
  - New highest bid → notify previous top bidder

Service-to-Service Communication

- 🧩 Patterns Used:
  - Sync (REST/gRPC):
    - User auth, listing fetch
    - Auction → Bid → Payment trigger
  - Async (Pub/Sub or Event Bus):
    - New bid placed → broadcast to watchers
    - Auction ended → notify winner + trigger payment
    - Failed payment → retry/alert
- 🕸️ Sample Event Topics:
  - auction.ended
  - bid.placed
  - payment.failed
  - user.registered

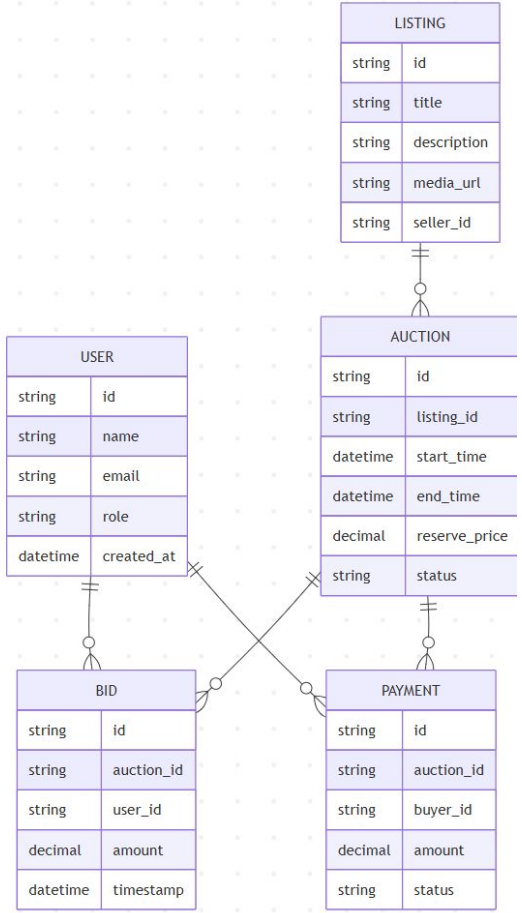
Benefits: Loosely coupled systems, retries, better failure handling

Real-Time Bid Delivery (WebSockets)

- Why WebSockets?
  - Low latency, full-duplex communication
  - Push updates to all watchers instantly
- How it works:
  - Client subscribes to auction channel: auction:{id}
  - Bid service validates & accepts bid → publishes event
  - WebSocket server fans out update to all connected clients
  - Key Concern: Horizontal scale under high concurrency

Data Model (Simplified)

- USER
  - Purpose: Stores information about the users (buyers/sellers)
- LISTING
  - Purpose: Contains details about the items being auctioned
- AUCTION
  - Purpose: Represents an auction for a specific listing, including its start and end time
- BID
  - Purpose: Stores bids placed on an auction by users
- PAYMENT
  - Purpose: Tracks payments after auction completion



Handling Auction Timers & Closures - Scheduled Job System for Auction Lifecycle

- Responsibilities:
  - Start auction at start_time
  - End auction at end_time
  - Determine winner, notify parties, trigger payment
- Design Options:
  - Dedicated Scheduler Service (cron/queue-based)
  - Use Delayed Jobs via message queue (e.g., SQS delay, Kafka, BullMQ)
  - Store timers in Redis + polling/expiration-based mechanism
- Reliability Measures:
  - Retry failed closures
  - Idempotent end-of-auction logic
  - Logging + alerts on missed timers

Strategic Tech & Infra Decisions for TinyURL

- Technology Stack for Scalability:
  - Frontend: React or Vue for responsive, real-time UI.
  - Backend: Node.js or Java with Spring Boot for handling API requests efficiently.
  - Database: PostgreSQL for relational data, Redis for caching high-demand data like auction info.
- Ensuring High Availability & Performance:
  - Load Balancing: Use cloud-based load balancers to distribute traffic evenly across servers.
  - Real-time Updates: WebSockets for real-time bidding and auction status updates.
  - Caching: Redis to cache frequently accessed data, improving performance and reducing database load.
- Security Considerations:
  - Authentication: OAuth2 for secure user login (buyers/sellers).
  - Data Encryption: SSL/TLS for encrypting data in transit, ensuring secure transactions.
- Cost Optimization:
  - Cloud Services: Use cloud providers' pay-as-you-go models to optimize infrastructure costs.
  - Autoscaling: Leverage cloud autoscaling to manage traffic surges, ensuring resource efficiency.

The Final Design - Auction Platform

- 🌐 API Gateway
- 👥 User Service
- 📦 Auction Service
- 🏷️ Listing Service
- ⚡ Bid Service
- 🧾 Payment Service
- 📣 Notification Service
- 🧭 Scheduler Service