Full Stack Developer – Case-Based Answers

ImpactHub Connect

---

1. Real-Time Audio Calls Integration (Twilio/Exotel)

Integration Strategy:

- Use Exotel or Twilio APIs to set up toll-free virtual call sessions.
- Assign each user a virtual number through the provider dashboard or programmatically.
- Backend coordinates the call setup, bridging both user numbers via a server-side integration.

Backend Changes Required:

- Implement WebSocket or long polling to update user status (Online, Busy).
- Save each call's metadata such as caller/receiver IDs, start and end timestamps, call status, and duration.
- Integrate coin deduction logic based on duration (e.g., 1 coin per 60 seconds).

Frontend Changes Required:

- Show live availability (Online/Busy) using real-time updates from backend.
- Call UI with buttons: “Start Call,” “End Call,” a timer, and call status.
- After the call, display the total duration and the number of coins deducted.

2. Wallet & KYC System

Wallet Schema (Example):

```
json
CopyEdit
{
  "userId": "string",
  "balance": 100,
  "transactions": [
    {
      "type": "credit",
      "amount": 50,
      "timestamp": "2025-07-01T10:00:00Z",
      "description": "Referral Bonus"
    },
    {
      "type": "debit",
      "amount": 10,
      "timestamp": "2025-07-01T11:00:00Z",
      "description": "Joined Game"
```

```
    }
  ],
  "referralCode": "ABHY123"
}
```

Referral Logic:

- Each user is assigned a unique referral code on sign-up.
- When a new user signs up using an existing code, both the referrer and referee receive a fixed coin bonus (e.g., 50 coins).
- To prevent misuse, validate the IP address, device ID, and email for uniqueness.

KYC Verification using API (Karza/Signzy):

- Integrate the Karza or Signzy SDK or API into the user onboarding flow.
- Allow document uploads (Aadhaar, PAN, Driving License).
- Show KYC verification status to users: `Pending`, `Verified`, or `Rejected`.
- Only users with `Verified` KYC status should be allowed to withdraw earned coins.

3. Performance & Scalability

Top 3 Priorities for Scaling to 1 Lakh+ Users:

1. **Backend Optimization:**
    - Use Node.js with clustering (PM2) and Redis for session storage.
    - Optimize database queries with indexes, caching, and pagination.
    - Use async queues (e.g., Bull) for processing heavy operations like video/audio generation or logs.
2. **CDN and Lazy Loading:**
    - Serve all static files through a CDN like Cloudflare or AWS CloudFront.
    - Lazy-load non-critical React components using `React.lazy` and `Suspense`.
3. **Scalable Database Architecture:**
    - Use MongoDB sharding or PostgreSQL partitioning.
    - Auto-scale read replicas in high-read situations.
    - Use connection pooling (e.g., pg-pool or mongoose-pool) for DB access.

Optimizations for Tier 2/3 Cities (Low-End Devices & Networks):

- Compress all static assets (WebP images, minified CSS/JS).
- Minimize JavaScript bundle size using tree shaking and code splitting.
- Use Service Workers (PWA pattern) to cache data and enable offline-first behavior.
- Defer loading of third-party scripts and ads until main content has loaded.
- Use Skeleton Loaders and Progressive Rendering for better UX.

4. Team Collaboration (Post-Agency Handover)

Key Audit Areas Before Taking Ownership:

1. **Code Quality & Structure:**

    - Review folder structure, modularity, component reusability, and consistency.
    - Identify redundant or duplicate code.
2. **API & Auth Handling:**
    - Validate all API contracts, request/response consistency, and error handling.
    - Review token/session management logic (JWT or OAuth).
3. **Infrastructure & Configuration:**
    - Secure access to all environment variables, deployment scripts, and third-party keys.
    - Understand CI/CD pipelines (GitHub Actions, Jenkins, etc.) and Docker usage.
4. **Documentation & Testing:**
    - Check for README files, API docs (Swagger/Postman), and setup instructions.
    - Review test coverage and run all unit/integration tests.

Handling Tech Debt:

- Categorize technical debt into immediate (e.g., security, scalability) and non-critical.
- Plan sprint-wise refactoring: begin with low-effort, high-impact items.
- Add missing unit tests for key modules and write developer-level documentation.
- Use ESLint, Prettier, and Git hooks (Husky) for clean code practices.
- Log all open issues in GitHub/JIRA to make refactoring traceable.

Summary

This case-based submission outlines how to architect and scale a real-world production-ready social app module using modern tools and APIs. The design ensures scalability, monetization logic, and developer collaboration best practices.