1. Real-Time Audio Calls:

1 How would you integrate a toll-free audio system like Exotel/Twilio?

> *Most of those uses socket i.e event based i.e connect and stream audio to other users. I have Similar project with vapi ai. if its just p2p then we should use webrtc. Its direct p2p just need to use small server to exchange packets.*

2 What backend and frontend changes are required to handle:

> *Socket based we need to use socket io. client side we need to handle events. logic is simple we can either directly send audio to other users if we have sdk or we need to send it to server and then server will handle it. use state and timer to handle call duration and coin deduction.*

2. Wallet & KYC System:

Outline how you would structure:

Wallet Database Schema

A robust wallet system requires two primary tables: one for the wallet itself and another for recording transactions.

`wallets` table:

- `wallet_id` *(Primary Key, e.g., UUID) - Unique identifier for the wallet.*
- `user_id` *(Foreign Key to `users` table, unique) - Links the wallet to a specific user.*
- `balance` *(Decimal/Numeric) - Stores the current balance with precision to avoid floating-point errors.*
- `currency` *(Varchar, e.g., "COINS") - The currency type stored in the wallet.*
- `created_at` *(Timestamp) - When the wallet was created.*
- `updated_at` *(Timestamp) - When the balance was last updated.*

`transactions` table:

- `transaction_id` *(Primary Key, e.g., UUID) - Unique identifier for the transaction.*
- `wallet_id` *(Foreign Key to `wallets` table) - Links the transaction to a wallet.*
- `amount` *(Decimal/Numeric) - The value of the transaction. Can be positive (credit) or negative (debit).*
- `type` *(Enum: `DEPOSIT`, `WITHDRAWAL`, `GAME_FEE`, `REWARD`) - The category of the transaction.*
- `status` *(Enum: `PENDING`, `COMPLETED`, `FAILED`) - The current state of the transaction.*
- `description` *(Text) - A human-readable summary of the transaction.*
- `reference_id` *(Varchar, optional) - For linking to external data, like a payment gateway ID or a game session ID.*
- `created_at` *(Timestamp) - When the transaction was initiated.*

Referral Reward Logic

Database Schema Changes

`users` table (additions):

- *`referral_code` (uuid , Unique) - A unique code automatically generated for each user.*
- *`referred_by_code` (uuid , Nullable) - The referral code the user entered when they signed up.*

`referrals` table (new):

- *`referral_id` (Primary Key) - Unique ID for the referral record.*
- *`referrer_user_id` (Foreign Key to `users`) - The user who owns the referral code.*
- *`referee_user_id` (Foreign Key to `users`) - The new user who signed up using the code.*
- *`status` (Enum: `PENDING`, `COMPLETED`) - Tracks whether the reward condition has been met.*
- *`created_at` (Timestamp) - When the referral was made.*

Logic Flow

1. ***Code Generation**: When a new user (User A) registers, generate a unique `referral_code` and save it to their record in the `users` table.*
2. ***Referral Tracking**: A new user (User B) signs up and enters User A's `referral_code`. This code is stored in User B's `referred_by_code` field.*
3. ***Record Creation**: A new entry is created in the `referrals` table linking User A (referrer) and User B (referee) with a `PENDING` status.*
4. ***Reward Trigger**: The system monitors for a specific action from User B (e.g., first deposit). When this occurs, a backend event is triggered.*
5. ***Reward Distribution**: The event handler finds the `PENDING` referral record for User B. It then:*
    - *Creates a `REFERRAL_REWARD` transaction for User A.*
    - *Updates User A's wallet balance.*
    - *Updates the referral record's status to `COMPLETED` to prevent duplicate rewards.*

KYC Verification Using an API (e.g., Karza/Signzy)

1. Database Schema Changes

`users` table (addition):

- `kyc_status` *(Enum:* `NOT_STARTED`*,* `PENDING`*,* `VERIFIED`*,* `REJECTED`*) - For quick, easy access to a user's verification status.*

`kyc_details` table (new):

- `kyc_id` *(Primary Key) - Unique ID for the verification attempt.*
- `user_id` *(Foreign Key to* `users`*).*
- `provider` *(Varchar) - The KYC service used (e.g., 'Karza').*
- `provider_reference_id` *(Varchar) - The unique ID from the KYC provider's system for this verification.*
- `status` *(Enum:* `PENDING`*,* `VERIFIED`*,* `REJECTED`*).*
- `rejection_reason` *(Text, Nullable) - Stores the reason for failure.*
- `submitted_at` *(Timestamp).*
- `last_updated_at` *(Timestamp).*

2. Backend and API Logic

1. ***Submit KYC Endpoint (`/api/kyc/submit`):***
    - *The frontend sends the user's details (e.g., PAN number) to this endpoint.*
    - *The backend securely calls the external KYC API (e.g., Karza).*
    - *It creates a record in* `kyc_details` *with a* `PENDING` *status and stores the* `provider_reference_id`*.*
    - *It updates the* `users.kyc_status` *to* `PENDING`*.*
2. ***Webhook Endpoint (`/api/kyc/webhook`):***
    - *The KYC provider sends asynchronous updates to this secure endpoint.*
    - *The backend verifies the request and uses the* `provider_reference_id` *to find the user.*
    - *It updates the* `status` *in both* `kyc_details` *and* `users` *to* `VERIFIED` *or* `REJECTED`*.*

3. Frontend Changes

- ***KYC Form**: A simple form in the user's profile to submit their information.*
- ***Status Display**: The UI must clearly show the user's current KYC status (`Not Verified`, `Pending`, `Verified`).*
- ***Error Handling**: If verification is rejected, the UI should display the reason and allow the user to resubmit.*

4. Performance + Scalability:

> *What would be your top 3 priorities when preparing Social App for 1 lakh+ daily users?*

- horizontal scaling we can use load balancer and use multiple servers.
- We can either write policy or use kubernetes to handle scaling.
- Here multiple things are required to handle like queues, Cache , load balancers, etc.

> *How would you ensure fast load times for users in Tier 2/3 cities with low-end devices?*

- We can use CDN to handle static files.
- Use centralize caching for all reads.
- Client side Optimisation like memoization, lazy loading, caching,etc.

5. Team Collaboration (Handover Scenario):

- What key areas would you audit before taking ownership from the agency?
    - Code quality
    - Documentation
    - Security
    - Scalability
    - Testing
    - Deployment
    - Monitoring
    - Maintenance
    - Environments
- How would you handle tech debt, documentation gaps, or refactoring if needed?
- AI can help a lot here.
- We need full docs & KT before handover.
- But let me be honest dont have mush idea here.