Website name is gamearena

muhe flly modern chiaye and bhout sare libraries ka use karna hai mujhe

Mujhe ek fully saas website create karna hai jisme abhi present mai 6 matches hoga

1. BGMI SOLO
2. BGMI DUO
3. BGMI SQUAD
4. FREE FIRE SOLO
5. FREE FIRE DUO
6. FREE FIRE SQUAD

AND ISME BHOUT TYPE KE TOURNAMENT HOGA JAISE KI SOLO DUO AND SQAUD

SOLO MEIN PER PERSON SE ₹20 LIYE JAENGE and jo match win ho jayega usko 350 rs milega and runner up ko 250 rs milega and per kill 9 rs milega

And duo mai per team 40 rs and winning price 350rs and runner up ko 250 and isme bhe same 9 rs per kill

And squad mai 80 rs per Team ka lagega and winning price 350 rs and runner up 250 milega and per kill 9rs

Pubg ke andar squad mein 25 teams a sakta hai

And duo mai 50 teams and solo mai 100 players

And yahi free fire mein squad mein 12 team Ja sakti hai and Duo mein 24 team and solo mai 48 players

Agar aap squire mein join karte hain to aapko 80rs per team dena padenge Duo mein join karte Hain 40rs per team and agar aap solo join karna chahte ho to aapko 20 rs Dena padega and ismein Jo winner price rahega 350 rahega for solo duo and squad and runner up price rahega 150 RS and per kill aapko ₹5 milega

And pictures ka bhi use karna to aur behtar rahega website ke liye

And website mai video bhe hina chaiye

AND HAAR EK FILES MAI COMMENTS HONA CHAIYE AMD EASY TO UNDERSTAND ENGLISH MAI COMMENTS LIKHA HUA HONA CHAIYE

AND SUPERBASE SE CONNECT RANHA CHAIYE MAI TUMKO SARA API KEY PROVIDE KAR DUNGA AND SUPERBASE MAI JITNA BHE CODE LIKHA HUA HONA CHAIYE YE SAB CHEEZ KA CODE HONA CHAIYE MATLAB KI SUPERCASE KE LIYE EK ALAG FOLDER 🗀 BANA LENA AND USI MAI CODE LIKHNA PURA

Matlab sare folder aacha se organise kar lena bgmi ke liye alag folder free fire ke liye alag

And ye sab ka jo registration form rahegaa wo bhe alag alag hee rahegaa

1. BGMI SOLO
2. BGMI DUO
3. BGMI SQUAD
4. FREE FIRE SOLO
5. FREE FIRE DUO
6. FREE FIRE SQUAD

And form mai agar koi name daal raha hai to sirf character hee valid rahegaa and mobile no mai sirf Indian no hee hee and

Baki details mai tumko niche provide kar de raha hu padh lena

Mujhe ek bhout hee jada professional and modern website create karna hai and uske andar like Jo Mera website rahega website mein isliye kar create kar raha hun ki usme mai tournament karba saku

BGMI AND FREE FIRE KA

AND ISME BHOUT TYPE KE TOURNAMENT HOGA JAISE KI SOLO DUO AND SQAUD

SOLO MEIN PER PERSON SE ₹20 LIYE JAENGE and jo match win ho jayega usko 350 rs milega and runner up ko 250 rs milega and per kill 9 rs milega

And duo mai per team 40 rs and winning price 350rs and runner up ko 250 and isme bhe same 9 rs per kill

And squad mai 80 rs per Team ka lagega and winning price 350 rs and runner up 250 milega and per kill 9rs

Pubg ke andar squad mein 25 teams a sakta hai

And duo mai 50 teams and solo mai 100 players

And yahi free fire mein squad mein 12 team Ja sakti hai and Duo mein 24 team and

solo mai 48 players

Agar aap squire mein join karte hain to aapko 80rs per team dena padenge Duo mein join karte Hain 40rs per team and agar aap solo join karna chahte ho to aapko 20 rs Dena padega and ismein Jo winner price rahega 350 rahega for solo duo and squad and runner up price rahega 150 RS and per kill aapko ₹5 milega

And pictures ka bhi use karna to aur behtar rahega website ke liye

And Tum to jante Hi Ho ki yah sab karne ke liye website ke andar do sections hone chahie do nahin sorry teen section hone chahie pahla jo hai homepage second pubg and third free fire home page ke andar thoda bahut detail rahana chahie pabji kya game hai and free fire kya game hai and jab Koi pubg wala page open Karega to usmein rul regulation rahana chahie likha hua pahle Se Hi ki iska kya rul Hai क्या-क्या policy ham log de rahe hain yah sab rahega and ek form rahega form mein team leader ke name rahega like sabse pahle team name rahega FIR team leader ka naam rahega kya rahega uska naam jo bhi hai uske bad pubg ka id 3 litre ka pubg ka id team leader ke WhatsApp ka number jo ki active Rahe hamesha and player tu ka name player tu ka pubg id player 3 ka name player three ke pubg ID player 4 ka name and player for ka pubg ID yah Ho Gaya squad match ke liye

Like pubg page per jab Koi open Karega to uske andar teen section baithe hue hone chahie first solo squad and duo
And same as a he duo ke liye bhe rahegaa idme tum khud se apna logic laga lena

And payment karne ke liye qr code pahle Se Hi rahega to agar aap payment kar dete hain to uska screenshot aapko dalna padega and transaction ID dalna padega uske andar aur aap submit kar sakte hain

Same aise hee free fire ke liye bhe page rahegaa similar iske jaise hee

And ismein bhi Tum apna logic and mind use kar lena

And your pubg pesh ke andar teen section rahega solo squad and duo abhi ke liye to yah rahega ki lagbhag sabke andar yahi show Karega plot Van karke like agar Main solve karestation open karta hun registration form ya FIR Jo Tum Bolo usko form bol lo to mere ko stall like yah admin ke pass jaega admin approve Karega to Aisa matlab sabji mein teen sections rahega to uska FIR Excel sheet aaega free fire mein teen section rahega uska FIR excel sheet to Matlab admin ko jo hai bahut problem sahana padega usko kyunki check bhi karna padega Koi agar transaction kar raha hai to sahi se transaction hua Hai ki nahin vah matlab screenshot dekh ke hi approve Karega to admin panel bahut hi humko advance banana padega iske liye taki sab chij sahi se pata chal paye like admin panel mein bhi do section rahega ek pubg ke liye aur ek free fire ke liye aur usmein hi matlab jo bhi upload karega jo bhi upload karega screenshot V uske andar dikhna chahie ki han Jaise pahla koi team Hai team ka name hoga to uska pura team ka details player ka niyam vagaira sab chij uske andar dikhna chahie payment Jo usne Kiya vah dekhna chahie aur uske sabse last mein

approve ya FIR reject ka option mere pass rahana chahie agar Main approve kar deta hun to 18 select ho gai hai aur reject kar deta hun to usmein kuchh gadbadi hai to Aisa Hi matlab iske andar bhi free fire ke andar bhi sem rahega but pubg mein bhi Aisa Hi rahega and teen stall har ek bar rahega pubg mein bhi aur free fire mein bhi

To like main yah bol raha hun ki Jaise slot Van pubg ke andar bhi teen aur section hone chahie admin page ke andar hi aise slot 1 solo slot 1 duo and slot 1 squad

Same free fire ke liye bhi sim logic Hi rahega

And mujhe Apne website ke liye back and front and sab chij tumhen Banakar Dena tumko Jo matlab Lage ki han yah language best rahega is website ke liye like backend mein bhi to bahut hai language use hote Hain

To Jo tumko Lage ki nahin yah jo hai language best rahega Jo Main website banaa raha hun uske according to Ve language to use kar lena backend ke liye and front and mein mere ko fulli professional and modern website chahie

And main ise Abhi pure acche tarike Se to host nahin karunga but main ise vercel par diploy karunga to usmein Matlab mere ko koi error nahin aana chahie jab Main diploy karne jaaun and Mera back hand bhi front and bhi sab chij acche se kam karna chahie

And ek aur baat Jaise matlab slot to pubg ke andar squire mein 25 hi team magistrate kar sakti hai and duo mein 50 team and solo mein 100 player

Tum mujhe isiliye time mein hamesha update chahie like agar Main Man ke chaliye vehicle per agar koi team register Karti Hai uske bad mein aur kisi ko bhi yah link bhejta Hun to agar koi team register kar chuki hai to uska matlab yah stall dikhna chahie ki han ek stall phool ho chuka hai plot nahin sorr
Matlab yah pata lag Jana chahie Jaise agar koi squire mein game khelna chahta Hai to Main squad mein ek Team register kar chuki hai to ab bacha 25 stall mein se ek stall Khali ho gaya hai to usko 24 plot hi dikhna chahie 24 stall available hai FIR agar vah bhi register kar leti Hai to 23 slot Khali rahana chahie aise hi solo mein agar koi ek player register kar leta hai to 99 stall Khali rahana chahie matlab 99 uske andar usko seat vacant dikhne chahie stall mein बार-बार bol dena vah galti Hai Mera but tum samajh jao apna logic use kar lena

And same logic free fire ke liye bhi rahega

Bhai abhi क्या-क्या problem a raha hai main tumko batata hun sabse pahle problem to a raha hai ki like jab Main review and submit page per jata Hun bgmi mein ya FIR free fire mein to yah matlab thoda lag hone lagta hai aur buffering type ka matlab show hone lagta hai jo ki mujhe aisa nahin chahie agar koi submit per click karen to uska submit Ho jaaye but aage jakar yah main check Karun ki han usne Jo screenshot

share kiya hai payment ka sahi hai ki nahin hai agar Sahi rahega to usko Whatsapp ke through main message kar dunga and Main usko share kar dunga room Idea and uska password

And bgmin and free fire agar aap Vin hote Hain aapane Koi kil Kiya hoga to aapko payment lene ke liye Jaise aap jite ho to aap payment loge hi to uske liye aapko jis Whatsapp ke through message jaega use per aapko last matlab match khatm hone ke bad aapko screenshot Dena padega

And mujhe ise vercel e deploy karna hai but problem yeh hai ki like Main agar refresh kar le raha hun to Jaise agar Main first time Maine Sara detail Bhar Diya to slot ek full to ho ja raha hai and admin page per bhi uska data show kar raha hai and my approve kar raha hun to approve bhi ho ja raha hai but Jaise hi main yah website aur kisi ke sath share kar raha hun and jab We open kar raha hai to usko slot pura ka pura Khali dikh raha hai jabki aisa nahin hona chahie jab ek stall Bhar Gaya Hai to bhara hua hi so hona chahie usko.

And admin page bhi बार-बार refresh nahin hona chahie but yah बार-बार ho ja raha hai बार-बार matlab sem problem mere ko face karna pad raha hai ki website refresh karte hi Sara data gayab ho ja raha hai and agar Main kisi ke sath share bhi kar raha hun to bhi Sara data gayab ho ja raha hai like vercel par deploy karne ke bad mein link to share kar sakta hun apni website ka to agar aisa hoga to sab Ko to fir बार-बार matlab plot Khali dikhega admin page bhi matlab refresh karne ke bad Sara data khatm Ho jaega to iska koi matlab nahin rahega na fir to yah bahut hi bekar website banaya Tumne

To acche se har ek page ko debug karo and jo jo maine bola hai usko Sahi karne ka TRAI karo

And admin page bhi tumhara acche se barf karna chahie fulli function mein hona chahie matlab jo bhi chij Tumne website ke andar Diya Hai sab kuchh fully functional mujhe chahie

And data jo hai free fire ka and bgmi ka donon ka अलग-अलग hi hona chahie Aisa nahin hona chahie ki donon ka sem Hi Ho jaaye

And ek aur problem a raha hai I like agar Main registration kar leta Hun and apna registration pura complete kar bhi leta Hun to fir Main agar Bgmi ke page per aata hun and free fire ke page per aata Hun to mere ko jo maine data feel kiya tha mere ko vahi data fir se show karta hai to Aisa agar rahega to main kisi ke sath agar link share karunga to yah to problem ki baat Ho jayegi Tum ek acche Pro developer ki tarah socho and soch samajhkar fulli logic and mind lagakar kam karo

Jo Tumne front and banaya Hai Jo Tumne Yuva UX banaya Hai Jo Tumne font use Kiya Hai Sara chij mast hai usko kuchh mat karna jo jo tumne front and mein kiya hai usko kuchh mat Karna and fulli logic and mind lagakar kuchh kuchh changes karo jo jo maine bola hai jo jo maine bola hai vah changes to hun nahin chahie

AND MUJHE FULLY MODERN WEBSITE CHAIYE AND FORMS BHE MODERN HEE HONA CHAIYE AND SAB CHEEZ WEBSITE MAI MODERN HEEE HONA CHAIYE JAISE FONTS TEXT COLORS
AND JITNA BHE MODERN WEBSITE HAI WORLD LEVEL PAR USKE JAISE HONA CHAIYE VERCEL JAISE MODERN HONA CHAIYE AND GITHUB JAISE AND SUPERBASE CURSOR apple

AND SARE PAGES MAI DETAILING HONA CHAIYE AND MODERN HONA CHAIYE AND BHOUT SARE LIBRARIES KA USE KARNA HAI

AND SARE PAGES MODERN HEE HONA CHAIYE

AND WEBSITE MAI TYPING SOUND BHE CHAIYE

VITE_SUPABASE_PROJECT_ID="ielwxcdoejxahmdsfznj"

VITE_ADMIN_EMAIL=ishukriitpatna@gmail.com
VITE_ADMIN_PASSWORD=ISHUkr75@

VITE_SUPABASE_URL=https://ielwxcdoejxahmdsfznj.supabase.co
VITE_SUPABASE_PUBLISHABLE_DEFAULT_KEY=sb_publishable_OOQjSct460-UZ6xSOJrmiw_JgfGPRuQ

GameArena — BGMI & Free Fire Tournament SaaS
Key Clarifications Needed
1) Stack choice: Keep existing Vite Ul or build fresh in Next.js 14 (App Router) on Vercel?
2) Payments: QR + screenshot only, slot reserved on submit (counts as filled), freed if rejected?
Architecture
• Frontend: Next-js 14 (App Router), TypeScript, TailwindCSS, shadcn/ui, React Hook Form + Zod, SWR Framer Motion.
Global typing sound on inputs.
• Backend: Supabase (Postgres + Auth + Storage + Realtime). Optional Edge Function for signed upload (phase 2).
• Persistence: All slot availability derived from DB (no client-only state). Realtime updates via Supabase Channels on
registrations .
• Deployment: Vercel. Env mapping for Supabase.
Directory Structure
•
•
•
•
•
•
— Homepage with images/videos, game intros, CTAs
app/(bgmi)/solo/page.tsx, duo/page.tsx, squad/page.tsx — BGMI forms
app/(freefire)/solo/page.tsx, dtn/page.tsx, — Free Fire forms
— Admin login; — overview
— filtered queues
— filtered queues
canponents/ — Ulf forms, media, sound, slot meters
and Iib/supabase/server.ts — browser/server clients
lib/validation/ — Zod schemas (names, Indian mobile, IDs)
— data access, RPC wrappers
public/assets/ — images, QR, demo videos
supabase/ — SOI migrations, seed, policies, types

Database Schema (Supabase SQL)
•
•
•
•
•
Enums: = (• bgmi', • freefire') , = (•solo', , • 41ad') , reg_status =
(• pending • , ' approved' , • rejected •)

tournaments (id, game, mode, fee_rs, prize_win_rs, prize_runner_rs, capacity)
registrations (id, toumament_id FK, status, team_name, leader_name, leader_game_id, leader_whatsapp,
transaction_id, payment_proof_path, created_at)
participants (id, registration_id FK slot_index, player_name, player_game_id)
(id, registration_id, admin_user_id, action, reason, created_at)
Indexes on and created_at for dashboard performance.
Capacities & Fees (Seed)
• BGMI: solo cap=100 (fee 20, win 350, runner 250, kill 9); duo cap=50 (team fee 40); squad cap=25 (team fee 80)
• Free Fire: solo cap=48 (fee 20, win 350, runner 150, kill 5); duo cap—24 (team fee 40); squad cap—12 (team fee 80)
Concurrency & Slot Logic
•
•
•
•
RPC function (SECURITY DEFINER):
Validates inputs and capacity atomically (FOR UPDATE on toumament row)
Inserts registrations (status= •pending') + participants
Returns current filled/remaining
Availability = count of registrations with status IN Rejection frees slot.
Realtime: client subscribes to registrations by tour-narnt_id to live-update remaining slots.
Storage & Media
• Bucket payment_proofs (public read). Policy: allow insert via RPC retuming signed upload URL OR direct client upload
with type/size validation. Start with direct client upload + file size/type checks; upgrade to signed uploads later.
• Homepage: hero video (lazy, muted, auto, poster), image carousel.

Forms & Validation
•
•
•
•
•
•
6 separate forms with tailored fields:
Solo: name, game_id, WhatsApp, transaction_id, screenshot
Duo: team_name, leader + mate name/game_id, WhatsApp, transaction_id, screenshot
Squad: team_name, leader + 3 players name/game_id, WhatsApp, transaction_id, screenshot
Zod: name letters-only, WhatsApp Indian transaction id non-empty, image type/size
Client image compression before upload; non-blocking UX (create row first, upload file next with progress)
Admin Panel
• Auth: Supabase email/password (use provided admin email). Add in user metadata.

• Dashboard: per game/mode tabs with counts (pending/approved/rejected, remaining slots) and search/filter.
• Row actions: view details, view/zoom proof, approve/reject (requires reason on reject). CSV export per list.
• Realtime updates to queues; no manual refresh. Persist selections via URL params.
Performance & UX
Lazy-loaded media, route-based code splitting, React Server Components where possible
• Avoid client-wide state; SWR + server components for data fetching
• Optimistic Ul on submit, toast notifications, loading skeletons
• Typing sound: lightweight audio sprite on input events; user toggle in footer
Security & RLS
•
•
•
•
•
RLS enabled on all tables
Policies:
tournarnts : read for all
registrations : anonymous can insert via RPC only; row read after creation by matching transaction_id (masked in
Ul); admin can read/update all
participants : same model as registrations
: admin-only
Admin role distinguished by user metadata claim; server checks in middleware.

Environment & Deployment
•
•
•
•
•
•
Map env for Next.js:
= provided URL
= provided publishable key
(server-only, for migrations/optional admin actions)
AD-IIN_EHAIL, (server-only bootstrap)
Vercel: set envs per project; build Next.js 14, image optimization enabled.
Deliverables
• Fully functional website with homepage, BGMI/FF pages, 6 modem forms, realtime slot meters, QR payment section,
image/video usage, typing sounds
• Admin panel with approvals, filtering, realtime updates, export to CSV
• Supabase schema, policies, seeds, and RPC in supabase/ folder
• All files with concise English comments
Migration & Seed (Essential SQL Snippets)

• Create enums, tables, indexes
• Insert 6 tournaments with capacities and economics
• Create RLS and policies
• RPC register_tean function with locking and validation
Testing
• Unit: validation schemas
• E2E: happy paths for 6 forms; capacity edge (over-cap submit); admin approve/reject; realtime slot change
Handoff
• README with setup steps, env vars, and Vercel deploy instructions
• Notes for tuming on signed uploads later

O
O
O
O
O
O
O
O
O
O
o
o
o
Initialize Next.js 14 app, Tailwind, shadcn/ui, base layout

Add SOI migrations for enums, tables, RLS, policies, seeds

Implement RPC register_team with capacity checks and locking

Wire Supabase client (server/browser) and Realtime subscriptions

Build homepage with hero video, images, CTAs, typing sound toggle

Implement BGMI solo/duo/squad forms with Zod, uploads, UX

Implement Free Fire solo/duo/squad forms with Zod, uploads, UX

Add live remaining-slot meters and availability badges

Add admin login using Supabase auth, role-based guard

Build admin dashboard lists, filters, approve/reject, CSV export

Add image compression, lazy media, code split, skeletons

Configure envs and deploy to Vercel, smoke test all flows

GameArena - Tournament SaaS Platform
Core Requirements:
6 Tournament Types:

BGMI: Solo (100 players), Duo (50 teams), Squad (25 teams)
Free Fire: Solo (48 players), Duo (24 teams), Squad (12 teams)
Pricing & Prizes:

BGMI: Solo ₹20/person, Duo ₹40/team, Squad ₹80/team
Winner: ₹350, Runner-up: ₹250, Per Kill: ₹9
Free Fire: Solo ₹20/person, Duo ₹40/team, Squad ₹80/team
Winner: ₹350, Runner-up: ₹150, Per Kill: ₹5
Key Features:

Real-time slot tracking (shows remaining slots live)
Payment via QR code + screenshot upload
Separate registration forms for each game mode
Admin panel for approving/rejecting registrations
Data persistence across refreshes and sharing
Modern UI with typing sounds, videos, images
WhatsApp integration for winner notifications
Critical Issues to Fix:
❌ Data Loss on Refresh - All data disappears when refreshing
❌ Data Loss When Sharing - Other users see empty slots
❌ No Real-time Updates - Slot availability not syncing
❌ Buffering on Submit - Lag when submitting forms
❌ Form Data Persistence - Previously entered data shows again
❌ Mixed Data - BGMI and Free Fire data getting mixed
Technical Stack:
Given your requirements for Vercel deployment, modern UI, and Supabase backend, I recommend:

Frontend: React + TypeScript + Vite (current stack)
UI: TailwindCSS + shadcn/ui (already installed) + Framer Motion
Forms: React Hook Form + Zod validation
Backend: Supabase (Database + Auth + Storage + Realtime)
State Management: TanStack Query (already installed)
Deployment: Vercel
Implementation Plan for GameArena Tournament Platform

Phase 1: Supabase Setup & Database Architecture (Foundation)
1.1 Connect to Supabase

Configure Supabase client with provided credentials
Set up environment variables properly for Vite
Create browser and server-side Supabase clients
1.2 Database Schema

```
-- Enums
CREATE TYPE game_type AS ENUM ('bgmi', 'freefire');
CREATE TYPE match_mode AS ENUM ('solo', 'duo', 'squad');
CREATE TYPE registration_status AS ENUM ('pending', 'approved', 'rejected');

-- Tournaments table (pre-configured 6 tournaments)
CREATE TABLE tournaments (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  game game_type NOT NULL,
  mode match_mode NOT NULL,
  entry_fee_rs INTEGER NOT NULL,
  prize_winner_rs INTEGER NOT NULL,
  prize_runner_rs INTEGER NOT NULL,
  prize_per_kill_rs INTEGER NOT NULL,
  max_capacity INTEGER NOT NULL,
  is_active BOOLEAN DEFAULT true,
  created_at TIMESTAMPTZ DEFAULT now(),
  UNIQUE(game, mode)
);

-- Registrations table
CREATE TABLE registrations (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  tournament_id UUID REFERENCES tournaments(id) NOT NULL,
  status registration_status DEFAULT 'pending',
  team_name TEXT,
  leader_name TEXT NOT NULL,
  leader_game_id TEXT NOT NULL,
  leader_whatsapp TEXT NOT NULL,
  transaction_id TEXT NOT NULL,
  payment_screenshot_url TEXT,
  created_at TIMESTAMPTZ DEFAULT now(),
  updated_at TIMESTAMPTZ DEFAULT now()
);

-- Participants table (for duo/squad members)
CREATE TABLE participants (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  registration_id UUID REFERENCES registrations(id) ON DELETE CASCADE,
  player_name TEXT NOT NULL,
  player_game_id TEXT NOT NULL,
```

```
  slot_position INTEGER NOT NULL
);

-- Admin actions log
CREATE TABLE admin_actions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  registration_id UUID REFERENCES registrations(id),
  admin_user_id UUID REFERENCES auth.users(id),
  action registration_status NOT NULL,
  reason TEXT,
  created_at TIMESTAMPTZ DEFAULT now()
);
```

1.3 Row Level Security (RLS) Policies

Enable RLS on all tables
Public read access to tournaments
Anonymous users can insert registrations
Admin users can update registration status
Create admin role system using user_roles table

1.4 Storage Bucket

Create payment_screenshots bucket
Set up proper RLS policies for uploads
Configure public access for viewing

1.5 Database Functions

```
-- Atomic registration with slot checking
CREATE FUNCTION register_for_tournament(...)
RETURNS JSON
LANGUAGE plpgsql
SECURITY DEFINER
-- Locks tournament row, checks capacity, inserts registration + participants

-- Get available slots real-time
CREATE FUNCTION get_available_slots(tournament_id UUID)
RETURNS INTEGER
-- Counts approved registrations vs capacity
```

Phase 2: Project Structure & Core Setup

2.1 Folder Organization

```
src/
├── components/
│   ├── ui/              # shadcn components (existing)
│   ├── forms/           # Registration forms
│   │   ├── BGMISoloForm.tsx
│   │   ├── BGMIDuoForm.tsx
│   │   ├── BGMISquadForm.tsx
│   │   ├── FreeFireSoloForm.tsx
│   │   ├── FreeFireDuoForm.tsx
```

```
│   │   └── FreeFireSquadForm.tsx
│   ├── admin/           # Admin components
│   │   ├── AdminDashboard.tsx
│   │   ├── RegistrationQueue.tsx
│   │   └── ApprovalModal.tsx
│   ├── shared/          # Reusable components
│   │   ├── SlotCounter.tsx
│   │   ├── PaymentQR.tsx
│   │   ├── TypingSoundToggle.tsx
│   │   └── VideoHero.tsx
│   └── layout/
│       ├── Navbar.tsx
│       └── Footer.tsx
├── pages/
│   ├── Index.tsx        # Homepage
│   ├── BGMIPage.tsx     # BGMI main page
│   ├── FreeFirePage.tsx # Free Fire main page
│   ├── admin/
│   │   ├── Login.tsx
│   │   └── Dashboard.tsx
│   └── NotFound.tsx
├── lib/
│   ├── supabase/
│   │   ├── client.ts    # Browser client
│   │   ├── server.ts    # Server client (if needed)
│   │   └── types.ts     # Generated types
│   ├── validation/
│   │   ├── schemas.ts   # Zod validation schemas
│   │   └── validators.ts
│   ├── hooks/
│   │   ├── useTypingSound.ts
│   │   ├── useSlotAvailability.ts
│   │   └── useRegistration.ts
│   └── utils/
│       ├── constants.ts # Tournament configs
│       └── formatting.ts
├── contexts/
│   └── AdminAuthContext.tsx
└── assets/
    ├── sounds/
    │   └── typing.mp3
    ├── images/
    │   └── qr-code.png
    └── videos/
        ├── bgmi-hero.mp4
        └── freefire-hero.mp4
```

2.2 Validation Schemas (Zod)

```
// Name validation: Only alphabets and spaces
```

```
const nameSchema = z.string()
  .regex(/^[A-Za-z\s]+$/, "Name must contain only letters")
  .min(2).max(50);

// Indian mobile validation: 10 digits starting with 6-9
const mobileSchema = z.string()
  .regex(/^[6-9]\d{9}$/, "Enter valid 10-digit Indian mobile number");

// Game ID validation
const gameIdSchema = z.string()
  .min(3).max(20).nonempty();

// Transaction ID validation
const transactionIdSchema = z.string()
  .min(10).max(50).nonempty();
```

Phase 3: Frontend Pages

3.1 Homepage (/)

Hero section with auto-playing muted video
Introduction to BGMI and Free Fire tournaments
Cards showing 6 tournament types with:
Entry fee
Prize pool
Available slots (real-time)
"Register Now" CTA buttons
Rules and regulations section
Modern animations with Framer Motion

3.2 BGMI Page (/bgmi)

Game introduction and overview
Rules and regulations section
Three tabs: Solo | Duo | Squad
Each tab shows:
Tournament details (fee, prizes, capacity)
Real-time slot availability counter
Registration form
Payment QR code section

3.3 Free Fire Page (/freefire)

Same structure as BGMI page
Separate data and forms
Different capacities and prize structures

3.4 Admin Login Page (/admin/login)

Email/password authentication
Supabase auth integration
Protected route with role validation

3.5 Admin Dashboard (/admin/dashboard)

Two main sections: BGMI | Free Fire

Each section has 3 sub-tabs: Solo | Duo | Squad
Registration queue showing:
Team/player details
Payment screenshot (zoomable)
Transaction ID
All participant details
Approve/Reject buttons with reason field
Real-time updates using Supabase Realtime
Filter by status: Pending | Approved | Rejected
Export to Excel functionality
Search functionality
Phase 4: Registration Forms
4.1 Solo Form (BGMI & Free Fire)

```
// Fields:
- Player Name (letters only)
- Game ID
- WhatsApp Number (Indian mobile validation)
- Payment Screenshot Upload
- Transaction ID
- Terms & Conditions checkbox
```

4.2 Duo Form (BGMI & Free Fire)

```
// Fields:
- Team Name
- Leader Name
- Leader Game ID
- Leader WhatsApp
- Player 2 Name
- Player 2 Game ID
- Payment Screenshot Upload
- Transaction ID
- Terms & Conditions checkbox
```

4.3 Squad Form (BGMI & Free Fire)

```
// Fields:
- Team Name
- Leader Name
- Leader Game ID
- Leader WhatsApp
- Player 2 Name & Game ID
- Player 3 Name & Game ID
- Player 4 Name & Game ID
- Payment Screenshot Upload
- Transaction ID
- Terms & Conditions checkbox
```

4.4 Form Features

Multi-step form with progress indicator
Real-time validation on each field
Image compression before upload
Loading states during submission
Success/error toast notifications
Slot availability check before final submission
No data persistence between sessions (fresh form each time)
Phase 5: Real-time Slot Management
5.1 Slot Counter Component

```
// Display format: "24/25 Slots Filled" or "24 Slots Available"
// Color coding:
// - Green: >50% available
// - Yellow: 20-50% available
// - Red: <20% available
// - Gray: Full
```

5.2 Supabase Realtime Subscription

```
// Subscribe to registration changes
// Update slot counts across all connected clients
// Prevent registration when capacity reached
// Show "FULL" badge when no slots available
```

5.3 Atomic Operations

Use database function for registration
Lock tournament row during registration check
Prevent race conditions
Transaction rollback on failure
Phase 6: Admin Panel Features
6.1 Authentication

Supabase Auth with email/password
Admin role stored in user_roles table
Protected routes with middleware
Session persistence
6.2 Dashboard Features

Filter registrations by:
Game (BGMI/Free Fire)
Mode (Solo/Duo/Squad)
Status (Pending/Approved/Rejected)
Search by:
Team name
Leader name
Transaction ID
WhatsApp number
6.3 Registration Review

View full registration details
Zoom payment screenshot
Approve action:
Marks status as 'approved'
Decrements available slots
Logs admin action
Reject action:
Requires reason input
Marks status as 'rejected'
Frees up slot
Logs admin action with reason
6.4 Export Functionality

Export filtered results to CSV/Excel
Include all registration and participant details
Separate exports for each tournament type
Phase 7: UI/UX Enhancements
7.1 Modern Design System

Use shadcn/ui components
Custom color scheme matching Vercel/GitHub/Supabase aesthetics
Smooth animations with Framer Motion
Glassmorphism effects
Gradient accents
Dark mode support (optional)
7.2 Typography & Fonts

Modern font stack (Inter, Geist, etc.)
Proper font weights and sizes
Responsive typography
7.3 Typing Sound Effect

Subtle keyboard typing sound on input focus
Global toggle in footer/settings
Preference saved in localStorage
Non-intrusive volume level
7.4 Media Integration

Hero videos (BGMI and Free Fire gameplay)
Tournament showcase images
Payment QR code image
Lazy loading for performance
Proper aspect ratios
7.5 Loading States

Skeleton loaders for data fetching
Spinners for form submissions
Progress bars for file uploads
Smooth transitions

Phase 8: Performance & Optimization
8.1 Code Splitting

Lazy load admin pages
Route-based code splitting
Dynamic imports for heavy components
8.2 Image Optimization

Compress images before upload
Use WebP format where possible
Implement lazy loading
Proper srcset for responsive images
8.3 Caching Strategy

TanStack Query for data caching
Stale-while-revalidate pattern
Optimistic updates for better UX
8.4 Bundle Optimization

Tree shaking
Remove unused dependencies
Minimize bundle size
Analyze with Vite bundle analyzer
Phase 9: Data Persistence & State Management
9.1 Database as Source of Truth

All slot availability from database
No client-side slot counting
Real-time synchronization via Supabase
9.2 State Management

TanStack Query for server state
React Context for auth state
Local storage only for UI preferences (typing sound, theme)
No localStorage for business logic
9.3 Session Handling

Clear form after successful submission
No form data persistence between sessions
Proper cleanup on component unmount
Phase 10: Security & Validation
10.1 Input Validation

Client-side validation with Zod
Server-side validation in database functions
SQL injection prevention
XSS protection
10.2 Authentication & Authorization

Admin role verification

Protected API routes
RLS policies on all tables
Secure admin credentials
10.3 File Upload Security

Validate file types (images only)
Limit file size (max 5MB)
Sanitize filenames
Secure storage bucket policies
Phase 11: Testing & Deployment
11.1 Testing

Test all 6 registration forms
Test slot availability real-time updates
Test admin approval/rejection flow
Test with multiple simultaneous users
Test data persistence across refreshes and shares
11.2 Vercel Deployment

```
# Environment variables needed:
VITE_SUPABASE_URL=https://ielwxcdoejxahmdsfznj.supabase.co
VITE_SUPABASE_ANON_KEY=<your-anon-key>
VITE_ADMIN_EMAIL=ishukriitpatna@gmail.com
VITE_ADMIN_PASSWORD=ISHUkr75@
```

11.3 Post-Deployment

Configure Supabase redirect URLs
Test payment screenshot uploads
Verify real-time updates work
Test admin panel functionality
Load testing for concurrent registrations
Phase 12: Documentation
12.1 Code Comments

English comments in all files
Explain complex logic
Document component props
API function documentation
12.2 README

Setup instructions
Environment variables guide
Deployment steps
Admin credentials
Tournament configuration
12.3 User Guide (Optional)

How to register for tournaments
Payment process

How to check registration status
WhatsApp notification format
This comprehensive plan addresses all your requirements and fixes all the critical issues you're facing. The key solutions are:

☑ No Data Loss - Everything stored in Supabase database
☑ Real-time Sync - Supabase Realtime for live slot updates
☑ Proper Separation - BGMI and Free Fire data completely separate
☑ No Form Persistence - Fresh forms for each new registration
☑ Atomic Operations - Database functions prevent race conditions
☑ Modern UI - shadcn/ui + Framer Motion + typing sounds
☑ Scalable - Proper architecture for future enhancements

🎮 GameArena Tournament Platform - Complete Implementation Plan
Current Issues Identified:
❌ No Supabase Integration - Need to set up database connectivity
❌ No Data Persistence - Currently blank app, need complete backend
❌ No Real-time Updates - Need Supabase Realtime subscriptions
❌ No Forms Exist Yet - Need to build all 6 registration forms
❌ No Admin Panel - Need complete admin system
📋 Complete Implementation Strategy for GameArena
Phase 1: Foundation Setup (Backend Infrastructure)
1.1 Supabase Connection & Configuration
Create Supabase client configuration in src/lib/supabase/client.ts
Set up environment variables properly for Vite
Create type-safe client instance with proper error handling
Test connection to ensure credentials work
1.2 Database Architecture (Supabase SQL Migrations)
Create the following tables in order:

```
-- Step 1: Create enums for type safety
CREATE TYPE game_type AS ENUM ('bgmi', 'freefire');
CREATE TYPE match_mode AS ENUM ('solo', 'duo', 'squad');
CREATE TYPE registration_status AS ENUM ('pending', 'approved', 'rejected');

-- Step 2: Tournaments table (6 pre-configured tournaments)
CREATE TABLE tournaments (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  game game_type NOT NULL,
  mode match_mode NOT NULL,
  entry_fee_rs INTEGER NOT NULL,
  prize_winner_rs INTEGER NOT NULL,
  prize_runner_rs INTEGER NOT NULL,
  prize_per_kill_rs INTEGER NOT NULL,
  max_capacity INTEGER NOT NULL,
  is_active BOOLEAN DEFAULT true,
  created_at TIMESTAMPTZ DEFAULT now(),
```

```
  UNIQUE(game, mode)
);

-- Step 3: Registrations table
CREATE TABLE registrations (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  tournament_id UUID REFERENCES tournaments(id) NOT NULL,
  status registration_status DEFAULT 'pending',
  team_name TEXT,
  leader_name TEXT NOT NULL,
  leader_game_id TEXT NOT NULL,
  leader_whatsapp TEXT NOT NULL,
  transaction_id TEXT NOT NULL,
  payment_screenshot_url TEXT,
  created_at TIMESTAMPTZ DEFAULT now(),
  updated_at TIMESTAMPTZ DEFAULT now()
);

-- Step 4: Participants table (for duo/squad members)
CREATE TABLE participants (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  registration_id UUID REFERENCES registrations(id) ON DELETE CASCADE,
  player_name TEXT NOT NULL,
  player_game_id TEXT NOT NULL,
  slot_position INTEGER NOT NULL,
  created_at TIMESTAMPTZ DEFAULT now()
);

-- Step 5: Admin actions log
CREATE TABLE admin_actions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  registration_id UUID REFERENCES registrations(id),
  admin_user_id UUID REFERENCES auth.users(id),
  action registration_status NOT NULL,
  reason TEXT,
  created_at TIMESTAMPTZ DEFAULT now()
);

-- Step 6: Create indexes for performance
CREATE INDEX idx_registrations_tournament ON registrations(tournament_id);
CREATE INDEX idx_registrations_status ON registrations(status);
CREATE INDEX idx_participants_registration ON participants(registration_id);
CREATE INDEX idx_admin_actions_registration ON admin_actions(registration_id);
Seed the 6 tournaments:


INSERT INTO tournaments (game, mode, entry_fee_rs, prize_winner_rs,
prize_runner_rs, prize_per_kill_rs, max_capacity) VALUES
('bgmi', 'solo', 20, 350, 250, 9, 100),
('bgmi', 'duo', 40, 350, 250, 9, 50),
```

```
('bgmi', 'squad', 80, 350, 250, 9, 25),
('freefire', 'solo', 20, 350, 150, 5, 48),
('freefire', 'duo', 40, 350, 150, 5, 24),
('freefire', 'squad', 80, 350, 150, 5, 12);
1.3 Row Level Security (RLS) Setup

-- Enable RLS on all tables
ALTER TABLE tournaments ENABLE ROW LEVEL SECURITY;
ALTER TABLE registrations ENABLE ROW LEVEL SECURITY;
ALTER TABLE participants ENABLE ROW LEVEL SECURITY;
ALTER TABLE admin_actions ENABLE ROW LEVEL SECURITY;

-- Public can read tournaments
CREATE POLICY "Anyone can view tournaments" ON tournaments
  FOR SELECT USING (true);

-- Anonymous users can insert registrations
CREATE POLICY "Anyone can register" ON registrations
  FOR INSERT WITH CHECK (true);

-- Users can view their own registrations
CREATE POLICY "View own registrations" ON registrations
  FOR SELECT USING (true);

-- Participants follow registration access
CREATE POLICY "Anyone can view participants" ON participants
  FOR SELECT USING (true);

-- Admin actions - only admins
CREATE POLICY "Admins can log actions" ON admin_actions
  FOR INSERT WITH CHECK (auth.uid() IS NOT NULL);
1.4 Admin Role System

-- Create app_role enum
CREATE TYPE app_role AS ENUM ('admin', 'user');

-- Create user_roles table
CREATE TABLE user_roles (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL,
  role app_role NOT NULL,
  created_at TIMESTAMPTZ DEFAULT now(),
  UNIQUE(user_id, role)
);

ALTER TABLE user_roles ENABLE ROW LEVEL SECURITY;

-- Security definer function to check admin role
CREATE OR REPLACE FUNCTION public.has_admin_role(_user_id UUID)
RETURNS BOOLEAN
```

```
LANGUAGE sql
STABLE
SECURITY DEFINER
SET search_path = public
AS $$
  SELECT EXISTS (
    SELECT 1
    FROM public.user_roles
    WHERE user_id = _user_id AND role = 'admin'
  );
$$;

-- Admin policies
CREATE POLICY "Admins can view all registrations" ON registrations
  FOR SELECT USING (public.has_admin_role(auth.uid()));

CREATE POLICY "Admins can update registration status" ON registrations
  FOR UPDATE USING (public.has_admin_role(auth.uid()));
1.5 Storage Bucket for Payment Screenshots

-- Create storage bucket via Supabase dashboard or SQL
INSERT INTO storage.buckets (id, name, public)
VALUES ('payment_screenshots', 'payment_screenshots', true);

-- RLS policy for uploads
CREATE POLICY "Anyone can upload payment screenshots"
ON storage.objects FOR INSERT
WITH CHECK (bucket_id = 'payment_screenshots');

-- RLS policy for viewing
CREATE POLICY "Anyone can view payment screenshots"
ON storage.objects FOR SELECT
USING (bucket_id = 'payment_screenshots');
1.6 Critical Database Function (Atomic Registration)

-- This function prevents race conditions and ensures slot availability
CREATE OR REPLACE FUNCTION register_for_tournament(
  p_tournament_id UUID,
  p_team_name TEXT,
  p_leader_name TEXT,
  p_leader_game_id TEXT,
  p_leader_whatsapp TEXT,
  p_transaction_id TEXT,
  p_payment_screenshot_url TEXT,
  p_participants JSONB
)
RETURNS JSON
LANGUAGE plpgsql
SECURITY DEFINER
AS $$
```

```
DECLARE
  v_tournament RECORD;
  v_current_count INTEGER;
  v_registration_id UUID;
  v_participant JSONB;
BEGIN
  -- Lock tournament row to prevent race conditions
  SELECT * INTO v_tournament
  FROM tournaments
  WHERE id = p_tournament_id
  FOR UPDATE;

  -- Check if tournament exists and is active
  IF NOT FOUND THEN
    RETURN json_build_object('success', false, 'error', 'Tournament not found');
  END IF;

  IF NOT v_tournament.is_active THEN
    RETURN json_build_object('success', false, 'error', 'Tournament is not
active');
  END IF;

  -- Count approved registrations
  SELECT COUNT(*) INTO v_current_count
  FROM registrations
  WHERE tournament_id = p_tournament_id
    AND status = 'approved';

  -- Check capacity
  IF v_current_count >= v_tournament.max_capacity THEN
    RETURN json_build_object('success', false, 'error', 'Tournament is full');
  END IF;

  -- Insert registration
  INSERT INTO registrations (
    tournament_id,
    team_name,
    leader_name,
    leader_game_id,
    leader_whatsapp,
    transaction_id,
    payment_screenshot_url
  ) VALUES (
    p_tournament_id,
    p_team_name,
    p_leader_name,
    p_leader_game_id,
    p_leader_whatsapp,
    p_transaction_id,
    p_payment_screenshot_url
```

```
  )
  RETURNING id INTO v_registration_id;

  -- Insert participants if provided
  IF p_participants IS NOT NULL THEN
    FOR v_participant IN SELECT * FROM jsonb_array_elements(p_participants)
    LOOP
      INSERT INTO participants (
        registration_id,
        player_name,
        player_game_id,
        slot_position
      ) VALUES (
        v_registration_id,
        v_participant->>'player_name',
        v_participant->>'player_game_id',
        (v_participant->>'slot_position')::INTEGER
      );
    END LOOP;
  END IF;

  RETURN json_build_object(
    'success', true,
    'registration_id', v_registration_id,
    'slots_remaining', v_tournament.max_capacity - v_current_count - 1
  );
END;
$$;
Phase 2: Project Structure & Core Files
2.1 Directory Structure
src/
├── components/
│   ├── ui/                    # shadcn components (existing)
│   ├── forms/                 # Registration forms
│   │   ├── BGMISoloForm.tsx
│   │   ├── BGMIDuoForm.tsx
│   │   ├── BGMISquadForm.tsx
│   │   ├── FreeFireSoloForm.tsx
│   │   ├── FreeFireDuoForm.tsx
│   │   └── FreeFireSquadForm.tsx
│   ├── admin/                 # Admin panel components
│   │   ├── AdminDashboard.tsx
│   │   ├── RegistrationQueue.tsx
│   │   ├── ApprovalModal.tsx
│   │   └── StatsCards.tsx
│   ├── shared/                # Reusable components
│   │   ├── SlotCounter.tsx
│   │   ├── TournamentCard.tsx
│   │   ├── PaymentSection.tsx
│   │   ├── TypingSoundToggle.tsx
```

```
│   │   └── VideoHero.tsx
│   └── layout/
│       ├── Navbar.tsx
│       ├── Footer.tsx
│       └── GamePageLayout.tsx
├── pages/
│   ├── Index.tsx              # Homepage
│   ├── BGMIPage.tsx           # BGMI tournaments
│   ├── FreeFirePage.tsx       # Free Fire tournaments
│   ├── admin/
│   │   ├── Login.tsx
│   │   └── Dashboard.tsx
│   └── NotFound.tsx
├── lib/
│   ├── supabase/
│   │   ├── client.ts          # Browser Supabase client
│   │   ├── types.ts           # Database types
│   │   └── queries.ts         # Reusable queries
│   ├── validation/
│   │   └── schemas.ts         # Zod validation schemas
│   ├── hooks/
│   │   ├── useTypingSound.ts
│   │   ├── useSlotAvailability.ts
│   │   ├── useRegistration.ts
│   │   └── useRealtimeSubscription.ts
│   ├── contexts/
│   │   └── AdminAuthContext.tsx
│   └── utils/
│       ├── constants.ts       # Tournament configs
│       └── helpers.ts
├── assets/
│   ├── sounds/
│   │   └── typing.mp3
│   ├── images/
│   │   ├── qr-code.png
│   │   ├── bgmi-hero.jpg
│   │   └── freefire-hero.jpg
│   └── videos/
│       ├── bgmi-hero.mp4
│       └── freefire-hero.mp4
└── types/
    └── index.ts               # Shared TypeScript types
```

2.2 Validation Schemas (Zod)

Create src/lib/validation/schemas.ts:

```
import { z } from 'zod';

// Name validation: Only alphabets and spaces
export const nameSchema = z.string()
```

```
  .trim()
  .regex(/^[A-Za-z\s]+$/, "Name must contain only letters and spaces")
  .min(2, "Name must be at least 2 characters")
  .max(50, "Name must be less than 50 characters");

// Indian mobile validation: 10 digits starting with 6-9
export const indianMobileSchema = z.string()
  .trim()
  .regex(/^[6-9]\d{9}$/, "Enter valid 10-digit Indian mobile number");

// Game ID validation
export const gameIdSchema = z.string()
  .trim()
  .min(3, "Game ID must be at least 3 characters")
  .max(20, "Game ID must be less than 20 characters")
  .nonempty("Game ID is required");

// Transaction ID validation
export const transactionIdSchema = z.string()
  .trim()
  .min(10, "Transaction ID must be at least 10 characters")
  .max(50, "Transaction ID must be less than 50 characters")
  .nonempty("Transaction ID is required");

// Team name validation
export const teamNameSchema = z.string()
  .trim()
  .min(3, "Team name must be at least 3 characters")
  .max(30, "Team name must be less than 30 characters");

// Solo form schema
export const soloFormSchema = z.object({
  player_name: nameSchema,
  game_id: gameIdSchema,
  whatsapp: indianMobileSchema,
  transaction_id: transactionIdSchema,
  screenshot: z.instanceof(File)
    .refine((file) => file.size <= 5 * 1024 * 1024, "File size must be less than
5MB")
    .refine(
      (file) => ['image/jpeg', 'image/png', 'image/jpg'].includes(file.type),
      "Only JPG, JPEG, and PNG files are allowed"
    ),
  terms: z.boolean().refine((val) => val === true, "You must agree to terms and
conditions")
});

// Duo form schema
export const duoFormSchema = z.object({
  team_name: teamNameSchema,
```

```
  leader_name: nameSchema,
  leader_game_id: gameIdSchema,
  leader_whatsapp: indianMobileSchema,
  player2_name: nameSchema,
  player2_game_id: gameIdSchema,
  transaction_id: transactionIdSchema,
  screenshot: z.instanceof(File)
    .refine((file) => file.size <= 5 * 1024 * 1024, "File size must be less than
5MB")
    .refine(
      (file) => ['image/jpeg', 'image/png', 'image/jpg'].includes(file.type),
      "Only JPG, JPEG, and PNG files are allowed"
    ),
  terms: z.boolean().refine((val) => val === true, "You must agree to terms and
conditions")
});

// Squad form schema
export const squadFormSchema = z.object({
  team_name: teamNameSchema,
  leader_name: nameSchema,
  leader_game_id: gameIdSchema,
  leader_whatsapp: indianMobileSchema,
  player2_name: nameSchema,
  player2_game_id: gameIdSchema,
  player3_name: nameSchema,
  player3_game_id: gameIdSchema,
  player4_name: nameSchema,
  player4_game_id: gameIdSchema,
  transaction_id: transactionIdSchema,
  screenshot: z.instanceof(File)
    .refine((file) => file.size <= 5 * 1024 * 1024, "File size must be less than
5MB")
    .refine(
      (file) => ['image/jpeg', 'image/png', 'image/jpg'].includes(file.type),
      "Only JPG, JPEG, and PNG files are allowed"
    ),
  terms: z.boolean().refine((val) => val === true, "You must agree to terms and
conditions")
});
Phase 3: Frontend Pages Implementation
3.1 Homepage (src/pages/Index.tsx)
Features:

Hero section with auto-playing muted video (BGMI or Free Fire gameplay)
Welcome section introducing GameArena
"What are BGMI and Free Fire?" sections with images
6 tournament cards showing:
Game logo/image
Tournament mode (Solo/Duo/Squad)
```

Entry fee
Prize breakdown (Winner, Runner-up, Per Kill)
Real-time slot availability (e.g., "24/25 Slots Available")
"Register Now" button
Rules and regulations section (expandable accordion)
Modern animations using Framer Motion
Footer with typing sound toggle
3.2 BGMI Page (src/pages/BGMIPage.tsx)
Layout:

Hero banner with BGMI branding
Game introduction and overview
Rules and regulations section (detailed)
Three tabs: Solo | Duo | Squad
Each Tab Contains:

Tournament details card:
Entry fee: ₹20/₹40/₹80
Winner prize: ₹350
Runner-up prize: ₹250
Per kill: ₹9
Max capacity: 100/50/25
Real-time slot counter (updates live via Supabase Realtime)
Registration form (specific to mode)
Payment QR code section with instructions
Terms and conditions checkbox
3.3 Free Fire Page (src/pages/FreeFirePage.tsx)
Same structure as BGMI page, but with:

Free Fire branding and colors
Different capacities: 48/24/12
Different per kill prize: ₹5
Runner-up prize: ₹150
Completely separate data from BGMI
3.4 Admin Login Page (src/pages/admin/Login.tsx)
Features:

Email input (pre-filled: ishukriitpatna@gmail.com)
Password input (secure, no visibility toggle initially)
"Login" button
Supabase Auth integration
Error handling for invalid credentials
Redirect to dashboard on success
Protected route - redirects if already logged in
3.5 Admin Dashboard (src/pages/admin/Dashboard.tsx)
Layout:

Top navbar with logout button
Two main sections: BGMI | Free Fire
Each section has 3 sub-tabs: Solo | Duo | Squad

Dashboard Features:

Stats cards showing:
Total registrations
Pending approvals
Approved count
Rejected count
Slots remaining
Registration queue table with columns:
Registration ID (short)
Team/Player name
Leader name
WhatsApp number
Transaction ID
Screenshot thumbnail (clickable to zoom)
Status badge (Pending/Approved/Rejected)
Actions (View Details, Approve, Reject)
Real-time updates using Supabase Realtime
Filter dropdown: All | Pending | Approved | Rejected
Search bar (by name, transaction ID, WhatsApp)
Export to CSV button
Pagination (20 records per page)
Approval Modal:

Full registration details
All participant names and game IDs
Payment screenshot (zoomable, full-screen option)
Transaction ID
Registration timestamp
Approve button (marks as approved, decrements slot)
Reject button (requires reason textarea, frees slot)
Phase 4: Registration Forms (6 Forms)
4.1 Form Structure (Multi-Step)
Step 1: Player/Team Details

Solo: Player name, Game ID
Duo: Team name, Leader + Player 2 details
Squad: Team name, Leader + 3 players details
Step 2: Contact & Payment

WhatsApp number (with Indian flag icon)
Transaction ID input
Payment screenshot upload (with preview)
QR code display with payment instructions
Step 3: Review & Submit

Summary of all entered data
Edit buttons for each section
Terms and conditions checkbox
Submit button

4.2 Form Features
Real-time validation on blur
Character restrictions: Name fields only accept letters and spaces
Number validation: WhatsApp field only accepts valid Indian numbers (10 digits, starting with 6-9)
Image compression before upload (compress to max 800KB)
File type validation: Only JPG, JPEG, PNG
Loading states during:
Image upload to Supabase Storage
Form submission
Slot availability check
Error handling:
Network errors
Slot full error
Duplicate registration check
Success state:
Confetti animation
Success message with registration ID
WhatsApp notification info
"Register Another Team" button
No data persistence: Form resets completely after submission or page refresh
4.3 Slot Availability Check (Before Submission)

```
// Before final submission, check slots in real-time
const checkSlotAvailability = async (tournamentId: string) => {
  const { data, error } = await supabase
    .from('registrations')
    .select('id', { count: 'exact', head: true })
    .eq('tournament_id', tournamentId)
    .eq('status', 'approved');

  const tournament = await getTournamentById(tournamentId);
  const slotsRemaining = tournament.max_capacity - (data?.length || 0);

  if (slotsRemaining <= 0) {
    throw new Error('Tournament is now full. Please try another tournament.');
  }

  return slotsRemaining;
};
```

Phase 5: Real-time Slot Management (Critical for Fixing Data Loss)
5.1 Slot Counter Component (src/components/shared/SlotCounter.tsx)
Display Logic:

Green: >50% slots available
Yellow: 20-50% slots available
Red: <20% slots available
Gray + "FULL": No slots available
Real-time Updates:

```
// Subscribe to registration changes for specific tournament
useEffect(() => {
  const channel = supabase
    .channel(`tournament:${tournamentId}`)
    .on(
      'postgres_changes',
      {
        event: '*',
        schema: 'public',
        table: 'registrations',
        filter: `tournament_id=eq.${tournamentId}`
      },
      (payload) => {
        // Refetch slot count when any registration changes
        refetchSlotCount();
      }
    )
    .subscribe();

  return () => {
    supabase.removeChannel(channel);
  };
}, [tournamentId]);
```

5.2 Database as Source of Truth (Fixing Refresh Issue)

Problem: Data disappears on refresh because it's stored in client state Solution: Always fetch from Supabase database

```
// Use TanStack Query for automatic caching and refetching
const { data: tournament, isLoading } = useQuery({
  queryKey: ['tournament', tournamentId],
  queryFn: async () => {
    const { data, error } = await supabase
      .from('tournaments')
      .select('*')
      .eq('id', tournamentId)
      .single();

    if (error) throw error;
    return data;
  },
  staleTime: 30000, // Cache for 30 seconds
  refetchOnWindowFocus: true, // Refetch when user returns to tab
});
```

5.3 Atomic Registration Function (Preventing Race Conditions)

Problem: Two users registering simultaneously for the last slot Solution: Use database function with row locking

```
// Call the database function instead of direct insert
const registerTeam = async (formData: FormData) => {
  const { data, error } = await supabase.rpc('register_for_tournament', {
    p_tournament_id: tournamentId,
    p_team_name: formData.team_name,
    p_leader_name: formData.leader_name,
    p_leader_game_id: formData.leader_game_id,
    p_leader_whatsapp: formData.leader_whatsapp,
    p_transaction_id: formData.transaction_id,
    p_payment_screenshot_url: screenshotUrl,
    p_participants: formData.participants // JSON array
  });

  if (error) throw error;

  if (!data.success) {
    throw new Error(data.error);
  }

  return data;
};
```

Phase 6: Admin Panel Implementation

6.1 Authentication System

Login Flow:

```
// Use Supabase Auth with email/password
const loginAdmin = async (email: string, password: string) => {
  const { data, error } = await supabase.auth.signInWithPassword({
    email,
    password
  });

  if (error) throw error;

  // Check if user has admin role
  const { data: roleData } = await supabase
    .from('user_roles')
    .select('role')
    .eq('user_id', data.user.id)
    .eq('role', 'admin')
    .single();

  if (!roleData) {
    await supabase.auth.signOut();
    throw new Error('Unauthorized: Admin access required');
  }

  return data;
};
```

Protected Route:

```
// Middleware to check admin status
const useAdminAuth = () => {
  const navigate = useNavigate();

  useEffect(() => {
    const checkAuth = async () => {
      const { data: { session } } = await supabase.auth.getSession();

      if (!session) {
        navigate('/admin/login');
        return;
      }

      const { data: roleData } = await supabase
        .from('user_roles')
        .select('role')
        .eq('user_id', session.user.id)
        .eq('role', 'admin')
        .single();

      if (!roleData) {
        await supabase.auth.signOut();
        navigate('/admin/login');
      }
    };

    checkAuth();
  }, [navigate]);
};
```

6.2 Dashboard Data Fetching with Real-time Updates

```
// Fetch registrations with real-time subscription
const { data: registrations, refetch } = useQuery({
  queryKey: ['registrations', game, mode, status],
  queryFn: async () => {
    let query = supabase
      .from('registrations')
      .select(`
        *,
        tournament:tournaments(*),
        participants(*)
      `)
      .order('created_at', { ascending: false });

    // Filter by game and mode
    query = query.eq('tournament.game', game);
    query = query.eq('tournament.mode', mode);
```

```
    // Filter by status if not 'all'
    if (status !== 'all') {
      query = query.eq('status', status);
    }

    const { data, error } = await query;
    if (error) throw error;
    return data;
  }
});

// Real-time subscription for new registrations
useEffect(() => {
  const channel = supabase
    .channel('admin-registrations')
    .on(
      'postgres_changes',
      {
        event: '*',
        schema: 'public',
        table: 'registrations'
      },
      () => {
        refetch(); // Refetch data when any change occurs
      }
    )
    .subscribe();

  return () => {
    supabase.removeChannel(channel);
  };
}, [refetch]);
```

6.3 Approval/Rejection Flow

```
// Approve registration
const approveRegistration = async (registrationId: string) => {
  const { error } = await supabase
    .from('registrations')
    .update({ status: 'approved', updated_at: new Date().toISOString() })
    .eq('id', registrationId);

  if (error) throw error;

  // Log admin action
  await supabase.from('admin_actions').insert({
    registration_id: registrationId,
    admin_user_id: (await supabase.auth.getUser()).data.user?.id,
    action: 'approved'
  });
```

```
  toast.success('Registration approved successfully');
};

// Reject registration
const rejectRegistration = async (registrationId: string, reason: string) => {
  if (!reason.trim()) {
    throw new Error('Rejection reason is required');
  }

  const { error } = await supabase
    .from('registrations')
    .update({ status: 'rejected', updated_at: new Date().toISOString() })
    .eq('id', registrationId);

  if (error) throw error;

  // Log admin action with reason
  await supabase.from('admin_actions').insert({
    registration_id: registrationId,
    admin_user_id: (await supabase.auth.getUser()).data.user?.id,
    action: 'rejected',
    reason
  });

  toast.success('Registration rejected');
};
```

6.4 Export to CSV

```
const exportToCSV = (registrations: Registration[]) => {
  const headers = [
    'Registration ID',
    'Team Name',
    'Leader Name',
    'Leader Game ID',
    'WhatsApp',
    'Transaction ID',
    'Status',
    'Players',
    'Created At'
  ];

  const rows = registrations.map(reg => [
    reg.id.slice(0, 8),
    reg.team_name || '-',
    reg.leader_name,
    reg.leader_game_id,
    reg.leader_whatsapp,
    reg.transaction_id,
    reg.status,
```

```
    reg.participants.map(p => `${p.player_name} (${p.player_game_id})`).join('; '),
    new Date(reg.created_at).toLocaleString()
  ]);

  const csv = [headers, ...rows]
    .map(row => row.map(cell => `"${cell}"`).join(','))
    .join('\n');

  const blob = new Blob([csv], { type: 'text/csv' });
  const url = window.URL.createObjectURL(blob);
  const a = document.createElement('a');
  a.href = url;
  a.download = `registrations-${game}-${mode}-${new Date().toISOString()}.csv`;
  a.click();
};
```

Phase 7: UI/UX Enhancements

7.1 Modern Design System

Color Scheme (Matching Vercel/GitHub/Supabase):

```
:root {
  --background: 0 0% 3%; /* Almost black */
  --foreground: 0 0% 98%; /* Almost white */
  --primary: 165 100% 45%; /* Teal/Cyan accent */
  --primary-foreground: 0 0% 100%;
  --secondary: 240 5% 10%; /* Dark gray */
  --accent: 165 70% 35%; /* Darker teal */
  --destructive: 0 84% 60%; /* Red for errors */
  --border: 240 5% 15%; /* Subtle borders */
  --card: 240 5% 6%; /* Card backgrounds */
  --muted: 240 5% 20%; /* Muted text */
}
```

Typography:

Font family: Inter or Geist (install via npm)
Headings: Bold, larger sizes
Body: Regular weight
Code/IDs: Monospace font

Glassmorphism Effects:

```
.glass-card {
  background: rgba(255, 255, 255, 0.05);
  backdrop-filter: blur(10px);
  border: 1px solid rgba(255, 255, 255, 0.1);
  border-radius: 12px;
}
```

7.2 Animations with Framer Motion

Page Transitions:

```
<motion.div
  initial={{ opacity: 0, y: 20 }}
  animate={{ opacity: 1, y: 0 }}
  exit={{ opacity: 0, y: -20 }}
  transition={{ duration: 0.3 }}
>
  {children}
</motion.div>
```

Card Hover Effects:

```
<motion.div
  whileHover={{ scale: 1.02, y: -5 }}
  whileTap={{ scale: 0.98 }}
  transition={{ type: 'spring', stiffness: 300 }}
>
  <TournamentCard />
</motion.div>
```

Slot Counter Animations:

```
<motion.div
  key={slotsRemaining}
  initial={{ scale: 1.2, opacity: 0 }}
  animate={{ scale: 1, opacity: 1 }}
  transition={{ type: 'spring' }}
>
  {slotsRemaining} slots remaining
</motion.div>
```

7.3 Typing Sound Effect Implementation:

```
// Hook: useTypingSound
const useTypingSound = () => {
  const [enabled, setEnabled] = useState(() => {
    return localStorage.getItem('typing-sound-enabled') !== 'false';
  });

  const audio = useRef<HTMLAudioElement | null>(null);

  useEffect(() => {
    audio.current = new Audio('/assets/sounds/typing.mp3');
    audio.current.volume = 0.3; // Subtle volume
  }, []);

  const playSound = useCallback(() => {
    if (enabled && audio.current) {
      audio.current.currentTime = 0;
```

```
      audio.current.play().catch(() => {
        // Ignore autoplay errors
      });
    }
  }, [enabled]);

  const toggle = () => {
    const newState = !enabled;
    setEnabled(newState);
    localStorage.setItem('typing-sound-enabled', String(newState));
  };

  return { enabled, playSound, toggle };
};

// Usage in input component
<Input
  onKeyDown={(e) => {
    if (e.key.length === 1) { // Only for character keys
      playSound();
    }
  }}
  {...otherProps}
/>
```

Toggle Button in Footer:

```
<Button
  variant="ghost"
  size="sm"
  onClick={toggleTypingSound}
>
  {typingSoundEnabled ? (
    <Volume2 className="h-4 w-4" />
  ) : (
    <VolumeX className="h-4 w-4" />
  )}
  Typing Sound
</Button>
```

7.4 Media Integration

Hero Video (Homepage):

```
<video
  autoPlay
  muted
  loop
  playsInline
  poster="/assets/images/bgmi-hero.jpg"
  className="w-full h-[500px] object-cover"
```

```
>
  <source src="/assets/videos/bgmi-hero.mp4" type="video/mp4" />
</video>
Payment QR Code:


<div className="flex flex-col items-center gap-4">
  <img
    src="/assets/images/qr-code.png"
    alt="Payment QR Code"
    className="w-64 h-64 border-2 border-primary rounded-lg"
  />
  <div className="text-center">
    <p className="font-semibold">Scan to pay ₹{entryFee}</p>
    <p className="text-sm text-muted-foreground">
      Upload screenshot after payment
    </p>
  </div>
</div>
Phase 8: Performance Optimization
8.1 Code Splitting

// Lazy load admin pages
const AdminDashboard = lazy(() => import('./pages/admin/Dashboard'));
const AdminLogin = lazy(() => import('./pages/admin/Login'));

// Use Suspense for loading state
<Suspense fallback={<LoadingSpinner />}>
  <AdminDashboard />
</Suspense>
8.2 Image Optimization

// Compress images before upload
import imageCompression from 'browser-image-compression';

const compressImage = async (file: File) => {
  const options = {
    maxSizeMB: 0.8, // 800KB
    maxWidthOrHeight: 1920,
    useWebWorker: true
  };

  const compressedFile = await imageCompression(file, options);
  return compressedFile;
};
8.3 TanStack Query Caching

// Configure query client for optimal caching
const queryClient = new QueryClient({
  defaultOptions: {
```

```
    queries: {
      staleTime: 30000, // 30 seconds
      cacheTime: 300000, // 5 minutes
      refetchOnWindowFocus: true,
      refetchOnReconnect: true,
      retry: 1
    }
  }
});
```

Phase 9: Security Measures

9.1 Input Sanitization

```
// Sanitize user inputs before submission
const sanitizeInput = (input: string): string => {
  return input.trim().replace(/[<>]/g, '');
};
```

9.2 File Upload Validation

```
// Server-side validation in edge function (optional)
const validateUpload = (file: File) => {
  const allowedTypes = ['image/jpeg', 'image/jpg', 'image/png'];
  const maxSize = 5 * 1024 * 1024; // 5MB

  if (!allowedTypes.includes(file.type)) {
    throw new Error('Invalid file type');
  }

  if (file.size > maxSize) {
    throw new Error('File too large');
  }
};
```

9.3 Rate Limiting (Future Enhancement)

```
// Implement rate limiting for registrations
// Use Supabase Edge Functions with Upstash Redis
// Limit to 3 registration attempts per IP per hour
```

Phase 10: Testing Strategy

10.1 Manual Testing Checklist

Registration Flow:

- Submit solo form for BGMI
- Submit duo form for Free Fire
- Submit squad form with max players
- Test with invalid name (numbers)
- Test with invalid WhatsApp (non-Indian)
- Test with invalid file type
- Test with large file (>5MB)
- Test slot availability check
- Test when tournament is full

Real-time Updates:

- Open two browser windows
- Register in window 1
- Verify slot count updates in window 2
- Refresh window 2, verify data persists
- Share link with another device, verify slots sync

Admin Panel:

- Login with admin credentials
- View pending registrations
- Approve a registration
- Reject a registration (with reason)
- Verify slot count decrements on approval
- Verify real-time updates work
- Export to CSV
- Filter by status
- Search by name

Data Separation:

- Register for BGMI solo
- Register for Free Fire solo
- Verify BGMI admin panel shows only BGMI
- Verify Free Fire admin panel shows only Free Fire

10.2 Edge Cases

Multiple users registering simultaneously for last slot
Network interruption during form submission
Browser refresh during multi-step form
Admin approving/rejecting same registration twice
Uploading corrupted image file

Phase 11: Deployment to Vercel

11.1 Environment Variables

Create .env.local:

```
VITE_SUPABASE_URL=https://ielwxcdoejxahmdsfznj.supabase.co
VITE_SUPABASE_ANON_KEY=<your-anon-key>
VITE_ADMIN_EMAIL=ishukriitpatna@gmail.com
```

Important: Do NOT store admin password in env variables. Use Supabase Auth only.

11.2 Vercel Configuration

Create vercel.json:

```
{
  "buildCommand": "npm run build",
  "outputDirectory": "dist",
  "framework": "vite",
  "rewrites": [
    {
      "source": "/(.*)",
```

```
      "destination": "/index.html"
    }
  ]
}
```

11.3 Deployment Steps

Push code to GitHub repository

Connect repository to Vercel

Add environment variables in Vercel dashboard

Deploy

Configure Supabase redirect URLs:

Add Vercel production URL to Supabase Auth allowed redirect URLs

Format: https://your-app.vercel.app/**

Phase 12: Post-Deployment Configuration

12.1 Supabase Auth Redirect URLs

In Supabase Dashboard → Authentication → URL Configuration:

Add: http://localhost:5173/** (development)

Add: https://your-app.vercel.app/** (production)

12.2 Create Admin User

Run this SQL in Supabase SQL Editor:

```
-- First, manually create admin user via Supabase Auth UI
-- Then assign admin role:

-- Get user ID from auth.users table
SELECT id FROM auth.users WHERE email = 'ishukriitpatna@gmail.com';

-- Insert admin role
INSERT INTO user_roles (user_id, role)
VALUES ('<user-id-from-above>', 'admin');
```

Phase 13: Documentation

13.1 Code Comments (English)

Every file should have:

```
/**
 * BGMISoloForm.tsx
 *
 * Registration form for BGMI Solo tournaments.
 * Handles player registration with validation, image upload,
 * and real-time slot availability checking.
 *
 * Features:
 * - Multi-step form with progress indicator
 * - Real-time validation using Zod
 * - Image compression before upload
 * - Atomic registration via database function
 */
```

13.2 README.md

Create comprehensive README with:

Project overview
Technology stack
Setup instructions
Environment variables
Database setup
Deployment guide
Admin access instructions
Troubleshooting section
Critical Features to Fix Data Loss Issues
☑ Issue 1: Data Loss on Refresh
Solution: All data stored in Supabase database, fetched on component mount using TanStack Query with caching.

☑ Issue 2: Data Loss When Sharing
Solution: No client-side state for business logic. All users fetch from same database source.

☑ Issue 3: No Real-time Updates
Solution: Supabase Realtime subscriptions on registrations table for all clients.

☑ Issue 4: Buffering on Submit
Solution:

Compress images before upload
Use optimistic UI updates
Show loading skeletons
Implement proper error boundaries
☑ Issue 5: Form Data Persistence
Solution: No localStorage for form data. Forms reset completely after submission or unmount.

☑ Issue 6: Mixed Data
Solution: Separate database queries filtered by game and mode. Each tournament has unique ID.

Final Deliverables
☑ Fully functional homepage with videos, images, and 6 tournament cards
☑ Separate BGMI and Free Fire pages with 3 modes each
☑ 6 modern registration forms with real-time validation
☑ Payment QR code section with screenshot upload
☑ Real-time slot availability counters (no data loss on refresh/sharing)
☑ Complete admin panel with approval/rejection workflow
☑ Real-time admin dashboard updates
☑ Export to CSV functionality
☑ Typing sound effect with toggle
☑ Modern UI with Framer Motion animations
☑ Supabase database with proper schema, RLS, and functions
☑ Complete separation of BGMI and Free Fire data

☑ All files with detailed English comments
☑ Deployed to Vercel with no errors
☑ Admin authentication system with role-based access

🎮 GameArena - Complete Implementation Plan
Project Overview
GameArena is a professional SaaS tournament platform for BGMI and Free Fire games with real-time slot management, payment processing, and admin approval system.

Phase 1: Supabase Backend Setup
1.1 Database Schema Creation
Tables to create:

tournaments - Stores 6 tournament configurations

Fields: id, game (enum: bgmi/freefire), mode (enum: solo/duo/squad), entry_fee_rs, prize_winner_rs, prize_runner_rs, prize_per_kill_rs, max_capacity, is_active
Pre-seed with 6 tournaments:
BGMI Solo (capacity: 100, fee: ₹20, winner: ₹350, runner: ₹250, kill: ₹9)
BGMI Duo (capacity: 50, fee: ₹40, same prizes)
BGMI Squad (capacity: 25, fee: ₹80, same prizes)
Free Fire Solo (capacity: 48, fee: ₹20, winner: ₹350, runner: ₹150, kill: ₹5)
Free Fire Duo (capacity: 24, fee: ₹40, same prizes)
Free Fire Squad (capacity: 12, fee: ₹80, same prizes)
registrations - Stores all tournament registrations

Fields: id, tournament_id (FK), status (pending/approved/rejected), team_name, leader_name, leader_game_id, leader_whatsapp, transaction_id, payment_screenshot_url, created_at, updated_at
participants - Stores team members for duo/squad

Fields: id, registration_id (FK), player_name, player_game_id, slot_position
admin_actions - Logs all admin approval/rejection actions

Fields: id, registration_id (FK), admin_user_id (FK to auth.users), action, reason, created_at
user_roles - Tracks admin users

Fields: id, user_id (FK to auth.users), role (enum: admin/user)
1.2 Row Level Security (RLS) Policies
Enable RLS on all tables

Public read access for tournaments
Anonymous insert for registrations (via RPC function)
Admin-only access for registration updates and admin_actions
Security definer function has_admin_role() for role checking
1.3 Storage Bucket
Create payment_screenshots bucket (public read)
Policies for anyone to upload, anyone to view
File validation: max 5MB, only image/jpeg, image/png
1.4 Critical Database Function (Prevents Race Conditions)
register_for_tournament() RPC function:

Uses FOR UPDATE lock on tournament row
Atomically checks capacity vs approved registrations
Inserts registration + participants in transaction
Returns success/error with slots remaining
Prevents double-booking when multiple users register simultaneously
Phase 2: Project Structure & Configuration
2.1 Directory Organization

```
src/
├── components/
│   ├── ui/                    # shadcn components (keep existing)
│   ├── forms/                 # 6 registration forms
│   │   ├── bgmi/
│   │   │   ├── SoloForm.tsx
│   │   │   ├── DuoForm.tsx
│   │   │   └── SquadForm.tsx
│   │   └── freefire/
│   │       ├── SoloForm.tsx
│   │       ├── DuoForm.tsx
│   │       └── SquadForm.tsx
│   ├── admin/                 # Admin panel components
│   │   ├── Dashboard.tsx
│   │   ├── RegistrationQueue.tsx
│   │   ├── ApprovalModal.tsx
│   │   └── StatsCards.tsx
│   ├── shared/                # Reusable components
│   │   ├── SlotCounter.tsx
│   │   ├── TournamentCard.tsx
│   │   ├── PaymentSection.tsx
│   │   ├── VideoHero.tsx
│   │   └── TypingSoundToggle.tsx
│   └── layout/
│       ├── Navbar.tsx
│       ├── Footer.tsx
│       └── PageHeader.tsx
├── pages/
│   ├── Index.tsx              # Homepage with all 6 tournaments
│   ├── BGMI.tsx               # BGMI page with 3 tabs
│   ├── FreeFire.tsx           # Free Fire page with 3 tabs
│   └── admin/
```

```
│       ├── Login.tsx
│       └── Dashboard.tsx
├── lib/
│   ├── supabase/
│   │   ├── client.ts          # Supabase browser client
│   │   ├── queries.ts         # Reusable query functions
│   │   └── types.ts           # Database types
│   ├── validation/
│   │   └── schemas.ts         # Zod validation schemas
│   ├── hooks/
│   │   ├── useTypingSound.ts
│   │   ├── useSlotAvailability.ts
│   │   ├── useRegistration.ts
│   │   └── useRealtimeSubscription.ts
│   └── contexts/
│       └── AdminAuthContext.tsx
├── assets/
│   ├── sounds/
│   │   └── typing.mp3
│   └── images/
│       ├── qr-payment.png     # Payment QR code
│       ├── bgmi-logo.png
│       └── freefire-logo.png
└── types/
    └── index.ts               # Shared TypeScript interfaces
```

2.2 Supabase Client Setup

src/lib/supabase/client.ts:

```
import { createClient } from '@supabase/supabase-js'

const supabaseUrl = import.meta.env.VITE_SUPABASE_URL
const supabaseAnonKey = import.meta.env.VITE_SUPABASE_ANON_KEY

export const supabase = createClient(supabaseUrl, supabaseAnonKey)
```

Environment variables:

```
VITE_SUPABASE_URL=https://ielwxcdoejxahmdsfznj.supabase.co
VITE_SUPABASE_ANON_KEY=[will be provided]
VITE_ADMIN_EMAIL=ishukriitpatna@gmail.com
VITE_ADMIN_PASSWORD=ISHUkr75@
```

2.3 Validation Schemas (Zod)

src/lib/validation/schemas.ts:

nameSchema - Only letters and spaces, 2-50 chars
indianMobileSchema - Exactly 10 digits, starts with 6-9
gameIdSchema - 3-20 chars, alphanumeric
transactionIdSchema - 10-50 chars, required
teamNameSchema - 3-30 chars for team names
soloFormSchema - Player name, game ID, WhatsApp, transaction ID, screenshot, terms

duoFormSchema - Team name, leader + player2 details, payment info
squadFormSchema - Team name, leader + 3 players details, payment info
Phase 3: Homepage Design (src/pages/Index.tsx)
Layout Structure:
Hero Section

Animated heading with gradient: "Welcome to GameArena"
Subheading: "India's Premier BGMI & Free Fire Tournament Platform"
Modern button "Explore Tournaments" (smooth scroll to tournaments)
Background: Subtle gradient with gaming theme
About Section

Two columns layout
Left: "What is BGMI?" with image and description
Right: "What is Free Fire?" with image and description
Modern cards with hover effects
Tournament Section

Heading: "Active Tournaments"
Grid layout (3 columns on desktop, responsive)
6 Tournament Cards:
Each card shows:

Game logo/badge (BGMI or Free Fire)
Mode badge (Solo/Duo/Squad)
Entry fee with ₹ icon
Prize breakdown:
Winner: ₹350 (trophy icon)
Runner-up: ₹250 (medal icon)
Per Kill: ₹9 or ₹5 (crosshair icon)
Real-time slot counter (e.g., "24/25 Slots Filled" or "24 Slots Available")
Color coding: Green (>50%), Yellow (20-50%), Red (<20%), Gray (Full)
"Register Now" button
Hover effect: Card lifts with shadow
Rules & Regulations Section

Accordion component from shadcn
Sections:
How to Register
Payment Process
Prize Distribution
Match Rules
WhatsApp Notification Info
Footer

Links: Home | BGMI | Free Fire | Admin
Typing sound toggle switch
Copyright notice
Key Features:
Modern font: Inter or Geist Sans

Color scheme: Dark background with vibrant accents (blue/purple gradients)
Framer Motion animations on scroll
Responsive design (mobile-first)
Fast load with lazy-loaded images
Phase 4: Game Pages (BGMI & Free Fire)
4.1 BGMI Page (src/pages/BGMI.tsx)
Structure:

Page Header

BGMI logo and branding
Gradient background with game theme
Breadcrumb: Home > BGMI
Game Overview Section

Brief description of BGMI
Key features of tournaments
Rules Section

Detailed tournament rules
Payment instructions with QR code
Prize distribution explanation
Registration process steps
Tournament Tabs (Solo | Duo | Squad)

Modern tab component from shadcn
Active tab highlighted with animation
Each tab contains:

A. Tournament Details Card

Entry Fee: ₹20/₹40/₹80
Max Teams/Players: 100/50/25
Winner Prize: ₹350
Runner-up Prize: ₹250
Per Kill: ₹9
Real-time slot counter with visual bar
B. Registration Form (Mode-specific)

Multi-step form with progress indicator
Step 1: Player/Team details
Step 2: Payment info & screenshot upload
Step 3: Review & submit
Real-time validation
Loading states
Success/error toasts
C. Payment Section

QR code image
Payment instructions

UPI details
Screenshot upload requirements
4.2 Free Fire Page (src/pages/FreeFire.tsx)
Same structure as BGMI page with differences:

Free Fire branding and colors
Different capacities: 48/24/12
Runner-up prize: ₹150
Per kill: ₹5
Completely separate data and forms
Phase 5: Registration Forms (6 Forms)
5.1 Form Architecture
All forms use:

React Hook Form for form management
Zod for validation
Multi-step wizard UI
TanStack Query for mutations
5.2 Solo Form Structure
Step 1: Player Details

- Player Name (text, letters only, 2-50 chars)
- Game ID (text, 3-20 chars)
- WhatsApp Number (10 digits, starts with 6-9, Indian flag icon)
Step 2: Payment

- Payment QR code display
- Transaction ID input (10-50 chars)
- Screenshot Upload (drag-drop or click, max 5MB, JPG/PNG only)
- Image preview after upload
- File size indicator
Step 3: Review & Submit

- Summary of all entered data
- Edit button for each section
- Terms and conditions checkbox
- Submit button with loading state
5.3 Duo Form Structure
Step 1: Team Details

- Team Name (3-30 chars)
- Leader Name (letters only)
- Leader Game ID
- Leader WhatsApp
- Player 2 Name (letters only)
- Player 2 Game ID
Step 2 & 3: Same as Solo

5.4 Squad Form Structure
Step 1: Team Details

- Team Name
- Leader Name
- Leader Game ID
- Leader WhatsApp
- Player 2 Name & Game ID
- Player 3 Name & Game ID
- Player 4 Name & Game ID

Step 2 & 3: Same as Solo

5.5 Form Submission Flow

Client-side validation (Zod schema)
Check slot availability (query registrations count)
Upload screenshot to Supabase Storage
Compress image to <800KB if needed
Generate unique filename
Upload to payment_screenshots bucket
Get public URL
Call register_for_tournament() RPC function
Pass all form data
Function handles capacity check atomically
Returns success + registration ID or error
Handle response:
Success: Show confetti, success message with registration ID, reset form
Error: Show error toast (e.g., "Tournament is full")
Refetch slot counts (triggers Realtime update for all clients)
5.6 Critical: No Form Data Persistence
Forms reset completely after successful submission
No localStorage for form data
No pre-filled data on page load
Fresh form for every new registration
Only typing sound preference stored in localStorage
Phase 6: Real-time Slot Management (Fixes Data Loss Issue)
6.1 Problem Analysis
Current issues:

Data disappears on refresh → Not stored in database
Slots show wrong count when sharing link → Client-side state only
Admin panel resets on refresh → No persistent data
Solution: Database as Single Source of Truth

6.2 Slot Counter Component
src/components/shared/SlotCounter.tsx:

```
// Fetches real-time slot count from database
const useSlotAvailability = (tournamentId: string) => {
  const { data, isLoading } = useQuery({
    queryKey: ['slots', tournamentId],
    queryFn: async () => {
```

```
      // Get tournament capacity
      const { data: tournament } = await supabase
        .from('tournaments')
        .select('max_capacity')
        .eq('id', tournamentId)
        .single()

      // Count approved registrations
      const { count } = await supabase
        .from('registrations')
        .select('*', { count: 'exact', head: true })
        .eq('tournament_id', tournamentId)
        .eq('status', 'approved')

      return {
        capacity: tournament.max_capacity,
        filled: count || 0,
        available: tournament.max_capacity - (count || 0)
      }
    },
    refetchInterval: 5000 // Refetch every 5 seconds as fallback
  })

  return data
}

// Display component
<div className="slot-counter">
  <div className="progress-bar" style={{ width: `${(filled/capacity)*100}%` }} />
  <span className={getColorClass(available, capacity)}>
    {available > 0 ? `${available} Slots Available` : 'FULL'}
  </span>
</div>
```

6.3 Supabase Realtime Subscription

src/lib/hooks/useRealtimeSubscription.ts:

```
// Subscribe to registration changes for live updates
useEffect(() => {
  const channel = supabase
    .channel(`tournament:${tournamentId}`)
    .on(
      'postgres_changes',
      {
        event: '*', // INSERT, UPDATE, DELETE
        schema: 'public',
        table: 'registrations',
        filter: `tournament_id=eq.${tournamentId}`
      },
      (payload) => {
```

```
        console.log('Registration change detected:', payload)
        // Invalidate and refetch slot count
        queryClient.invalidateQueries(['slots', tournamentId])
      }
    )
    .subscribe()

  return () => {
    supabase.removeChannel(channel)
  }
}, [tournamentId])
```

6.4 Benefits

☑ Slot counts update instantly across all connected clients
☑ Data persists across refreshes
☑ Link sharing shows correct slot counts
☑ Prevents race conditions with atomic database function
☑ Admin panel always shows current data

Phase 7: Admin Panel

7.1 Admin Login (src/pages/admin/Login.tsx)

Features:

Modern login card with gradient background
Email input (can pre-fill with provided email)
Password input (type="password", no visibility toggle initially)
"Login" button with loading state
Supabase Auth integration
Error handling (invalid credentials, network errors)
Redirect to dashboard on success
Protected: Redirects if already logged in

Authentication Flow:

```
const handleLogin = async (email: string, password: string) => {
  const { data, error } = await supabase.auth.signInWithPassword({
    email,
    password
  })

  if (error) throw error

  // Check if user has admin role
  const { data: role } = await supabase
    .from('user_roles')
    .select('role')
    .eq('user_id', data.user.id)
    .single()

  if (role?.role !== 'admin') {
    throw new Error('Unauthorized: Admin access required')
  }
```

```
  navigate('/admin/dashboard')
}
```

7.2 Admin Dashboard (src/pages/admin/Dashboard.tsx)

Layout:

Top Navigation Bar

GameArena logo
"Admin Dashboard" heading
User email display
Logout button
Main Tabs: BGMI | Free Fire

Modern tab switcher
Each game has 3 sub-tabs: Solo | Duo | Squad
Stats Cards Row (for selected tournament)

Total Registrations (large number with icon)
Pending Approvals (yellow badge)
Approved (green badge)
Rejected (red badge)
Slots Remaining (with progress bar)
Filters & Actions Row

Status filter dropdown: All | Pending | Approved | Rejected
Search input: Search by name, transaction ID, WhatsApp
Export CSV button
Refresh button (with auto-refresh indicator)
Registration Queue Table

Columns:

Sr. No.
Registration ID (short UUID, copyable)
Team/Player Name
Leader Name
WhatsApp (formatted: +91 XXXXX-XXXXX)
Game IDs (expandable for squad)
Transaction ID (copyable)
Screenshot (thumbnail, click to zoom)
Status (badge: Pending/Approved/Rejected)
Registered At (formatted date/time)
Actions (View Details button)
Row Actions:

Click anywhere: Open approval modal
Hover: Highlight row
Pagination
20 records per page

Page numbers + Previous/Next buttons
7.3 Approval Modal
Modal Layout:

Header:

"Registration Details" title
Close button (X)
Content Sections:

Registration Info

Registration ID
Submitted on: [date/time]
Status badge
Team/Player Details (Card)

Team Name (if duo/squad)
Leader/Player Name
Leader Game ID
Leader WhatsApp (with copy button)
All participants (for duo/squad):
Player 2, 3, 4 names and game IDs
Formatted as table
Payment Details (Card)

Transaction ID (large, copyable)
Payment Screenshot:
Large preview
Zoom in/out controls
Full-screen button
Download button
Admin Actions

If status = 'pending':

Large "Approve" button (green)
"Reject" button (red) → Opens reason textarea
If status = 'approved':

Green checkmark "Already Approved"
Approved by: Admin email
Approved at: Date/time
If status = 'rejected':

Red X "Rejected"
Rejection reason displayed
Rejected by: Admin email
Rejected at: Date/time
Rejection Flow:

```
Click "Reject" button
Textarea appears: "Enter rejection reason (required)"
Min 10 characters validation
Confirm Reject button
On submit:
Updates registration status to 'rejected'
Logs action in admin_actions table
Slot becomes available again
Shows success toast
Modal closes
Table refreshes
Approval Flow:

Click "Approve" button
Confirmation dialog: "Confirm approval?"
On confirm:
Updates registration status to 'approved'
Logs action in admin_actions table
Slot counter decrements
Shows success toast
Modal closes
Table refreshes
7.4 Real-time Updates in Admin Panel

// Subscribe to all registration changes for current tournament
useEffect(() => {
  const channel = supabase
    .channel(`admin:${selectedTournamentId}`)
    .on(
      'postgres_changes',
      {
        event: '*',
        schema: 'public',
        table: 'registrations',
        filter: `tournament_id=eq.${selectedTournamentId}`
      },
      (payload) => {
        // Refetch registrations and stats
        queryClient.invalidateQueries(['registrations', selectedTournamentId])
        queryClient.invalidateQueries(['stats', selectedTournamentId])
      }
    )
    .subscribe()

  return () => supabase.removeChannel(channel)
}, [selectedTournamentId])
7.5 Export to CSV Feature

const exportToCSV = (registrations: Registration[]) => {
```

```
  const headers = [
    'Registration ID',
    'Team Name',
    'Leader Name',
    'Leader Game ID',
    'WhatsApp',
    'Transaction ID',
    'Status',
    'Registered At',
    'Participants'
  ]

  const rows = registrations.map(reg => [
    reg.id,
    reg.team_name || '-',
    reg.leader_name,
    reg.leader_game_id,
    reg.leader_whatsapp,
    reg.transaction_id,
    reg.status,
    formatDate(reg.created_at),
    reg.participants.map(p => `${p.player_name} (${p.player_game_id})`).join('; ')
  ])

  // Generate CSV and download
  const csv = [headers, ...rows].map(row => row.join(',')).join('\n')
  const blob = new Blob([csv], { type: 'text/csv' })
  const url = URL.createObjectURL(blob)
  const a = document.createElement('a')
  a.href = url
  a.download = `registrations-${Date.now()}.csv`
  a.click()
}
Phase 8: UI/UX Enhancements
8.1 Modern Design System
Color Scheme:

Background: Dark navy (#0a0e27)
Primary: Electric blue (#3b82f6)
Secondary: Purple (#8b5cf6)
Accent: Cyan (#06b6d4)
Success: Green (#10b981)
Warning: Yellow (#f59e0b)
Danger: Red (#ef4444)
Text: White (#ffffff) and Gray (#94a3b8)
Components:

All shadcn/ui components styled consistently
Glassmorphism effects (backdrop-blur, semi-transparent backgrounds)
Gradient accents on cards and buttons
```

Smooth hover transitions (transform: translateY(-4px))
Box shadows for depth
8.2 Typography
Font Stack:

```
font-family: 'Inter', 'Geist Sans', system-ui, -apple-system, sans-serif;
```

Sizes:

Headings: 2xl-4xl with font-bold
Body: base with font-normal
Small text: sm with text-muted-foreground
Button text: sm with font-semibold
8.3 Animations (Framer Motion)
Homepage Hero:

```
<motion.h1
  initial={{ opacity: 0, y: 20 }}
  animate={{ opacity: 1, y: 0 }}
  transition={{ duration: 0.5 }}
>
  Welcome to GameArena
</motion.h1>
```

Tournament Cards:

```
<motion.div
  whileHover={{ scale: 1.05, y: -8 }}
  whileTap={{ scale: 0.95 }}
  className="tournament-card"
>
  {/* Card content */}
</motion.div>
```

Form Steps:

```
<motion.div
  initial={{ opacity: 0, x: 20 }}
  animate={{ opacity: 1, x: 0 }}
  exit={{ opacity: 0, x: -20 }}
  transition={{ duration: 0.3 }}
>
  {/* Step content */}
</motion.div>
```

Success Confetti:

```
import confetti from 'canvas-confetti'
```

```
const celebrateSuccess = () => {
  confetti({
    particleCount: 100,
    spread: 70,
    origin: { y: 0.6 }
  })
}
```

8.4 Typing Sound Effect

Implementation:

Audio File: src/assets/sounds/typing.mp3 (short, subtle click sound)

Hook: src/lib/hooks/useTypingSound.ts

```
export const useTypingSound = () => {
  const [enabled, setEnabled] = useState(() => {
    return localStorage.getItem('typingSoundEnabled') === 'true'
  })

  const audio = useRef<HTMLAudioElement>()

  useEffect(() => {
    audio.current = new Audio('/src/assets/sounds/typing.mp3')
    audio.current.volume = 0.2
  }, [])

  const playSound = () => {
    if (enabled && audio.current) {
      audio.current.currentTime = 0
      audio.current.play()
    }
  }

  const toggle = () => {
    const newState = !enabled
    setEnabled(newState)
    localStorage.setItem('typingSoundEnabled', String(newState))
  }

  return { enabled, playSound, toggle }
}
```

Usage in Forms:

```
const { playSound } = useTypingSound()

<input
  onKeyDown={playSound}
  {...register('player_name')}
/>
```

Toggle in Footer:

<TypingSoundToggle />
// Shows: "Typing Sound: [ON/OFF]" with switch
8.5 Loading States
Skeleton Loaders:

Tournament cards while fetching
Registration table rows while loading
Form submission with spinner overlay
Progress Indicators:

File upload progress bar
Multi-step form progress (e.g., "Step 2 of 3")
Spinners:

Button loading states (spinner replaces button text)
Full-page loading for protected routes
Phase 9: Performance Optimizations
9.1 Code Splitting

```
// Lazy load admin pages
const AdminDashboard = lazy(() => import('./pages/admin/Dashboard'))
const AdminLogin = lazy(() => import('./pages/admin/Login'))

// Wrap with Suspense
<Suspense fallback={<LoadingSpinner />}>
  <AdminDashboard />
</Suspense>
```

9.2 Image Optimization
Compress payment screenshots before upload (use browser-image-compression library)
Lazy load images with loading="lazy"
Use WebP format where supported
Provide proper width/height attributes
9.3 TanStack Query Configuration

```
const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      staleTime: 30000, // 30 seconds
      cacheTime: 300000, // 5 minutes
      refetchOnWindowFocus: false,
      retry: 1
    }
  }
})
```

9.4 Bundle Size Optimization
Tree-shake unused shadcn components
Use dynamic imports for heavy libraries (e.g., confetti)
Minimize dependencies

Phase 10: Security & Validation
10.1 Input Validation
Client-side (Zod):

Validates on blur and on submit
Shows error messages inline
Server-side (Database):

RPC function validates all inputs
SQL injection prevention (parameterized queries)
XSS protection (sanitize inputs)
10.2 File Upload Security

```
// Client-side validation
const validateFile = (file: File) => {
  const allowedTypes = ['image/jpeg', 'image/png', 'image/jpg']
  const maxSize = 5 * 1024 * 1024 // 5MB

  if (!allowedTypes.includes(file.type)) {
    throw new Error('Only JPG and PNG files allowed')
  }

  if (file.size > maxSize) {
    throw new Error('File size must be less than 5MB')
  }
}

// Compress before upload
const compressImage = async (file: File) => {
  const imageCompression = await import('browser-image-compression')
  return await imageCompression.default(file, {
    maxSizeMB: 0.8,
    maxWidthOrHeight: 1920,
    useWebWorker: true
  })
}
```

10.3 Authentication & Authorization
Admin Protection:

```
// Protected route wrapper
const ProtectedRoute = ({ children }) => {
  const { data: session, isLoading } = useQuery({
    queryKey: ['session'],
    queryFn: async () => {
      const { data } = await supabase.auth.getSession()
      return data.session
    }
  })
```

```
  const { data: role } = useQuery({
    queryKey: ['role', session?.user?.id],
    queryFn: async () => {
      const { data } = await supabase
        .from('user_roles')
        .select('role')
        .eq('user_id', session?.user?.id)
        .single()
      return data?.role
    },
    enabled: !!session
  })

  if (isLoading) return <LoadingSpinner />
  if (!session || role !== 'admin') {
    return <Navigate to="/admin/login" />
  }

  return children
}
```

10.4 RLS Policies (Recap)

All tables have RLS enabled

Public can only insert registrations via RPC (prevents direct manipulation)

Admin role checked via security definer function

Storage bucket policies restrict file types and sizes

Phase 11: Testing & Deployment

11.1 Manual Testing Checklist

Registration Flow:

- [ ] Test all 6 forms (BGMI + Free Fire, Solo + Duo + Squad)
- [ ] Validate name fields (reject numbers/special chars)
- [ ] Validate WhatsApp (only Indian numbers)
- [ ] Test file upload (reject >5MB, wrong file types)
- [ ] Test slot availability check (try when full)
- [ ] Verify data persists after refresh
- [ ] Verify slot counter updates in real-time

Multi-user Testing:

- [ ] Open in 2+ browsers
- [ ] Submit registration in one, verify slot updates in other
- [ ] Test race condition: submit simultaneously, only one should succeed if last slot

Admin Panel:

- [ ] Login with admin credentials
- [ ] View all 6 tournament queues
- [ ] Filter by status (Pending/Approved/Rejected)
- [ ] Search by name, transaction ID
- [ ] Approve a registration, verify slot decrements
- [ ] Reject a registration with reason, verify slot frees up
- [ ] Export CSV, verify data correctness

 Verify real-time updates when registration submitted
Data Persistence:

 Register → Refresh page → Verify slots persist
 Register → Share link → Verify slots correct on other device
 Admin approve → Refresh → Verify status persists
 Admin reject → Refresh → Verify slot available again
11.2 Vercel Deployment
Step 1: Connect GitHub

Push code to GitHub repository
Connect Vercel to GitHub
Step 2: Configure Environment Variables in Vercel

VITE_SUPABASE_URL=https://ielwxcdoejxahmdsfznj.supabase.co
VITE_SUPABASE_ANON_KEY=[anon key from Supabase dashboard]
VITE_ADMIN_EMAIL=ishukriitpatna@gmail.com
VITE_ADMIN_PASSWORD=ISHUkr75@
Step 3: Build Settings

Framework Preset: Vite
Build Command: npm run build
Output Directory: dist
Install Command: npm install
Step 4: Deploy

Click "Deploy"
Wait for build to complete
Test deployment URL
Step 5: Custom Domain (Optional)

Add custom domain in Vercel settings
Configure DNS records
11.3 Post-Deployment
Supabase Configuration:

Add Vercel domain to Supabase allowed redirect URLs
Test authentication flow on production
Verify file uploads work
Check Realtime subscriptions connect
Performance Check:

Run Lighthouse audit
Verify fast load times
Check mobile responsiveness
Phase 12: Documentation & Comments
12.1 Code Comments (English)
Every file includes:

```
/**
 * src/components/forms/bgmi/SoloForm.tsx
 *
 * Registration form for BGMI Solo tournaments.
 * Uses multi-step wizard with validation.
 * Submits data to Supabase via RPC function.
 */

// Import statements with comments explaining purpose
import { useForm } from 'react-hook-form' // Form state management
import { zodResolver } from '@hookform/resolvers/zod' // Zod integration

// Component with JSDoc
/**
 * BGMISoloForm - Handles BGMI Solo tournament registration
 * @param tournamentId - UUID of the tournament to register for
 */
export const BGMISoloForm = ({ tournamentId }: Props) => {
  // State for multi-step form (1: details, 2: payment, 3: review)
  const [step, setStep] = useState(1)

  // Form setup with Zod validation
  const form = useForm({
    resolver: zodResolver(soloFormSchema),
    defaultValues: {
      player_name: '',
      game_id: '',
      whatsapp: '',
      transaction_id: '',
      screenshot: null,
      terms: false
    }
  })

  // Handle form submission
  const onSubmit = async (data) => {
    // Step 1: Check slot availability
    const slotsAvailable = await checkSlots(tournamentId)
    if (slotsAvailable <= 0) {
      toast.error('Tournament is full!')
      return
    }

    // Step 2: Compress and upload screenshot
    const compressedFile = await compressImage(data.screenshot)
    const screenshotUrl = await uploadScreenshot(compressedFile)

    // Step 3: Register via RPC function (atomic operation)
    const { data: result, error } = await supabase.rpc(
      'register_for_tournament',
```

```
      {
        p_tournament_id: tournamentId,
        p_team_name: null, // Solo has no team name
        p_leader_name: data.player_name,
        p_leader_game_id: data.game_id,
        p_leader_whatsapp: data.whatsapp,
        p_transaction_id: data.transaction_id,
        p_payment_screenshot_url: screenshotUrl,
        p_participants: null // Solo has no additional participants
      }
    )

    if (error) {
      toast.error(error.message)
      return
    }

    // Step 4: Show success message
    celebrateSuccess()
    toast.success(`Registration successful! ID: ${result.registration_id}`)

    // Step 5: Reset form
    form.reset()
    setStep(1)
  }

  // Return JSX...
}
```

12.2 README.md

```
# 🎮 GameArena - Tournament Platform

Professional BGMI and Free Fire tournament management platform with real-time
updates.

## Features

- ☑ 6 Tournament Types (BGMI & Free Fire: Solo, Duo, Squad)
- ☑ Real-time Slot Management
- ☑ Payment Screenshot Upload
- ☑ Admin Approval System
- ☑ Multi-step Registration Forms
- ☑ Modern UI with Animations
- ☑ Typing Sound Effects
- ☑ Mobile Responsive

## Tech Stack

- **Frontend:** React 18, TypeScript, Vite
- **UI:** Tailwind CSS, shadcn/ui, Framer Motion
```

- **Backend:** Supabase (PostgreSQL, Auth, Storage, Realtime)
- **Forms:** React Hook Form, Zod
- **State:** TanStack Query

Setup Instructions

1. Clone Repository
```bash
git clone <repo-url>
cd gamearena
npm install
2. Configure Environment Variables
Create .env file:

VITE_SUPABASE_URL=https://ielwxcdoejxahmdsfznj.supabase.co
VITE_SUPABASE_ANON_KEY=<your-anon-key>
VITE_ADMIN_EMAIL=ishukriitpatna@gmail.com
VITE_ADMIN_PASSWORD=ISHUkr75@
3. Set Up Supabase Database
Run the SQL migrations in supabase/migrations/ in order:

001_create_tables.sql
002_create_rls_policies.sql
003_create_functions.sql
004_seed_tournaments.sql
Create storage bucket:

Name: payment_screenshots
Public: Yes
Apply policies from supabase/storage/policies.sql
4. Create Admin User
In Supabase dashboard:

Go to Authentication > Users
Create new user with email: ishukriitpatna@gmail.com
Run SQL:

INSERT INTO user_roles (user_id, role)
VALUES ((SELECT id FROM auth.users WHERE email = 'ishukriitpatna@gmail.com'),
'admin');
5. Run Development Server

npm run dev
Visit http://localhost:5173

Deployment
Vercel
Push code to GitHub
Import project in Vercel
Add environment variables
```

Deploy
Post-Deployment
Update Supabase redirect URLs with production domain
Test all features on production
Monitor Supabase usage
Admin Access
URL: /admin/login Email: ishukriitpatna@gmail.com Password: ISHUkr75@

Project Structure

```
src/
├── components/
│   ├── forms/           # 6 registration forms
│   ├── admin/           # Admin panel components
│   ├── shared/          # Reusable components
│   └── layout/          # Navbar, Footer
├── pages/
│   ├── Index.tsx        # Homepage
│   ├── BGMI.tsx         # BGMI tournaments
│   ├── FreeFire.tsx     # Free Fire tournaments
│   └── admin/           # Admin pages
├── lib/
│   ├── supabase/        # Database client & queries
│   ├── validation/      # Zod schemas
│   └── hooks/           # Custom React hooks
└── assets/              # Images, sounds
```

Tournament Configuration

Game	Mode	Capacity	Fee	Winner	Runner-up	Per Kill
BGMI	Solo	100	₹20	₹350	₹250	₹9
BGMI	Duo	50	₹40	₹350	₹250	₹9
BGMI	Squad	25	₹80	₹350	₹250	₹9
Free Fire	Solo	48	₹20	₹350	₹150	₹5
Free Fire	Duo	24	₹40	₹350	₹150	₹5
Free Fire	Squad	12	₹80	₹350	₹150	₹5

License
Proprietary - All rights reserved

12.3 Supabase Folder Structure

```
supabase/ ├── migrations/ │ ├── 001_create_tables.sql │ ├──
002_create_rls_policies.sql │ ├── 003_create_functions.sql │ └──
004_seed_tournaments.sql ├── storage/ │ └── policies.sql └── README.md
```

Phase 13: Key Differences from Current Issues

❌ Current Problems → ☑ Solutions

1. **Data disappears on refresh**
   - ❌ Problem: Using client-side state only
   - ☑ Solution: All data in Supabase database, TanStack Query for caching

2. **Slots show wrong count when sharing**
   - ❌ Problem: Each client has independent state
   - ☑ Solution: Database as source of truth + Realtime subscriptions

3. **Admin panel resets**
   - ❌ Problem: No persistent storage
   - ☑ Solution: Supabase Auth + database queries

4. **Form data persists between users**
   - ❌ Problem: localStorage or global state
   - ☑ Solution: Fresh form on every load, no persistence

5. **Race conditions on slot filling**
   - ❌ Problem: No atomic operations
   - ☑ Solution: Database function with FOR UPDATE lock

6. **BGMI and Free Fire data mixing**
   - ❌ Problem: Shared state or table
   - ☑ Solution: Separate by tournament_id, proper filtering

Phase 14: Additional Libraries to Install

```bash
# Core dependencies (already installed)
npm install @tanstack/react-query react-hook-form @hookform/resolvers zod

# Supabase
npm install @supabase/supabase-js

# Framer Motion for animations
npm install framer-motion

# Image compression
npm install browser-image-compression

# Confetti for success animation
npm install canvas-confetti
npm install --save-dev @types/canvas-confetti

# Date formatting
npm install date-fns

# CSV export
npm install papaparse
```

```
npm install --save-dev @types/papaparse

# Optional: Better table component
npm install @tanstack/react-table
```

🎮 GameArena Tournament Platform - Complete Implementation Plan
📋 Overview
Building a fully modern SaaS platform for BGMI & Free Fire tournament management with real-time slot tracking, payment processing, and comprehensive admin controls.

🎯 Core Requirements Summary
Tournament Types (6 Total)
BGMI:

Solo: 100 players, ₹20 entry, Winner ₹350, Runner-up ₹250, ₹9/kill
Duo: 50 teams, ₹40 entry, Winner ₹350, Runner-up ₹250, ₹9/kill
Squad: 25 teams, ₹80 entry, Winner ₹350, Runner-up ₹250, ₹9/kill
Free Fire:

Solo: 48 players, ₹20 entry, Winner ₹350, Runner-up ₹150, ₹5/kill
Duo: 24 teams, ₹40 entry, Winner ₹350, Runner-up ₹150, ₹5/kill
Squad: 12 teams, ₹80 entry, Winner ₹350, Runner-up ₹150, ₹5/kill
Key Features
☑ Real-time slot availability updates ☑ QR code payment + screenshot verification ☑ Separate organized forms for each mode ☑ Admin approval system with filtering ☑ Data persistence across sessions and users ☑ Modern UI with animations, typing sounds, videos ☑ Indian mobile validation, character-only names ☑ No data mixing between BGMI/Free Fire

🏗 Architecture & Modern Tech Stack
Frontend Framework
Vite + React 18 (current setup maintained)
TypeScript for type safety
TailwindCSS for styling
React Router v6 for routing
Modern UI Libraries
shadcn/ui (already installed) - Base components
Framer Motion - Smooth animations and page transitions
React Spring - Advanced spring physics animations
Lucide React (already installed) - Modern icons
React Confetti - Celebration effects on successful registration
React Hot Toast (sonner already installed) - Beautiful notifications
React Loading Skeleton - Elegant loading states
React Countup - Animated number counters for stats
Form & Validation Libraries
React Hook Form (already installed) - Performant forms

Zod (already installed) - Schema validation
@hookform/resolvers (already installed) - Integration layer
State Management & Data Fetching
TanStack Query v5 (already installed) - Server state management
Zustand - Lightweight client state (for UI preferences, sound toggle)
Supabase Realtime - Live slot updates
Media & Effects
use-sound - Typing sound effects library
react-player - Video player for homepage
react-lazy-load-image-component - Optimized image loading
sharp (via Vite) - Image optimization
Backend (Supabase)
PostgreSQL - Main database
Supabase Auth - Admin authentication
Supabase Storage - Payment screenshot storage
Supabase Realtime - Live updates
Row Level Security (RLS) - Data security
🗀 Project Structure

```
gamearena/
├── src/
│   ├── assets/              # Images, videos, sounds
│   │   ├── images/
│   │   ├── videos/
│   │   └── sounds/
│   │       └── typing.mp3
│   ├── components/
│   │   ├── common/          # Shared components
│   │   │   ├── Navbar.tsx
│   │   │   ├── Footer.tsx
│   │   │   ├── LoadingState.tsx
│   │   │   └── ErrorBoundary.tsx
│   │   ├── bgmi/            # BGMI specific components
│   │   │   ├── SoloForm.tsx
│   │   │   ├── DuoForm.tsx
│   │   │   ├── SquadForm.tsx
│   │   │   └── SlotMeter.tsx
│   │   ├── freefire/        # Free Fire specific components
│   │   │   ├── SoloForm.tsx
│   │   │   ├── DuoForm.tsx
│   │   │   ├── SquadForm.tsx
│   │   │   └── SlotMeter.tsx
│   │   ├── admin/           # Admin panel components
│   │   │   ├── AdminLayout.tsx
│   │   │   ├── RegistrationCard.tsx
│   │   │   ├── FilterControls.tsx
│   │   │   ├── ImageZoomModal.tsx
│   │   │   └── ExportCSV.tsx
│   │   └── ui/              # shadcn/ui components (existing)
│   ├── lib/
│   │   ├── supabase/
```

```
│   │   │   ├── client.ts     # Browser client
│   │   │   ├── server.ts     # Server-side client
│   │   │   ├── hooks.ts      # Custom hooks
│   │   │   └── types.ts      # Generated types
│   │   ├── validation/
│   │   │   ├── bgmi.ts       # BGMI form schemas
│   │   │   ├── freefire.ts   # Free Fire form schemas
│   │   │   └── common.ts     # Shared validators
│   │   ├── hooks/
│   │   │   ├── useSound.ts
│   │   │   ├── useSlotTracking.ts
│   │   │   └── useRegistration.ts
│   │   ├── stores/
│   │   │   └── uiStore.ts    # Zustand store
│   │   └── utils.ts
│   ├── pages/
│   │   ├── Index.tsx         # Homepage
│   │   ├── bgmi/
│   │   │   ├── BgmiLayout.tsx
│   │   │   ├── Solo.tsx
│   │   │   ├── Duo.tsx
│   │   │   └── Squad.tsx
│   │   ├── freefire/
│   │   │   ├── FreefireLayout.tsx
│   │   │   ├── Solo.tsx
│   │   │   ├── Duo.tsx
│   │   │   └── Squad.tsx
│   │   └── admin/
│   │       ├── Login.tsx
│   │       ├── Dashboard.tsx
│   │       ├── BgmiRegistrations.tsx
│   │       └── FreefireRegistrations.tsx
│   ├── styles/
│   │   └── animations.css    # Custom animations
│   └── App.tsx
├── supabase/
│   ├── migrations/
│   │   ├── 001_create_enums.sql
│   │   ├── 002_create_tables.sql
│   │   ├── 003_create_rls_policies.sql
│   │   ├── 004_create_functions.sql
│   │   └── 005_seed_tournaments.sql
│   ├── types.ts              # Generated TypeScript types
│   └── README.md             # Supabase setup guide
├── public/
│   └── qr-code.png          # Payment QR code
└── package.json
```

🗄️ Database Schema (Supabase)

Enums

```
CREATE TYPE game_type AS ENUM ('bgmi', 'freefire');
CREATE TYPE match_mode AS ENUM ('solo', 'duo', 'squad');
CREATE TYPE registration_status AS ENUM ('pending', 'approved', 'rejected');
```

Tables

1. tournaments - Pre-configured 6 tournament types

```
id (UUID, PK)
game (game_type)
mode (match_mode)
entry_fee_rs (INTEGER)
prize_winner_rs (INTEGER)
prize_runner_rs (INTEGER)
prize_per_kill_rs (INTEGER)
max_capacity (INTEGER)
is_active (BOOLEAN)
created_at (TIMESTAMPTZ)
```

UNIQUE constraint on (game, mode)

2. registrations - Main registration records

```
id (UUID, PK)
tournament_id (UUID, FK)
status (registration_status, default 'pending')
team_name (TEXT, nullable for solo)
leader_name (TEXT)
leader_game_id (TEXT)
leader_whatsapp (TEXT)
transaction_id (TEXT)
payment_screenshot_url (TEXT)
created_at (TIMESTAMPTZ)
updated_at (TIMESTAMPTZ)
```

3. participants - Additional team members (duo/squad)

```
id (UUID, PK)
registration_id (UUID, FK, ON DELETE CASCADE)
player_name (TEXT)
player_game_id (TEXT)
slot_position (INTEGER) - 1 for player 2, 2 for player 3, etc.
```

4. admin_actions - Audit log

```
id (UUID, PK)
registration_id (UUID, FK)
admin_user_id (UUID, FK to auth.users)
action (registration_status)
reason (TEXT, nullable)
created_at (TIMESTAMPTZ)
```

5. Storage Bucket: payment_screenshots

```
Public read access
Authenticated upload
Max file size: 5MB
```

Allowed types: image/jpeg, image/png
Database Functions
register_for_tournament - Atomic registration with slot checking

```
CREATE OR REPLACE FUNCTION register_for_tournament(
  p_tournament_id UUID,
  p_team_name TEXT,
  p_leader_name TEXT,
  p_leader_game_id TEXT,
  p_leader_whatsapp TEXT,
  p_transaction_id TEXT,
  p_payment_screenshot_url TEXT,
  p_participants JSONB
) RETURNS JSON AS $$
DECLARE
  v_capacity INTEGER;
  v_filled_slots INTEGER;
  v_registration_id UUID;
BEGIN
  -- Lock the tournament row for capacity check
  SELECT max_capacity INTO v_capacity
  FROM tournaments
  WHERE id = p_tournament_id AND is_active = true
  FOR UPDATE;

  IF v_capacity IS NULL THEN
    RETURN json_build_object('success', false, 'error', 'Tournament not found');
  END IF;

  -- Count approved + pending registrations
  SELECT COUNT(*) INTO v_filled_slots
  FROM registrations
  WHERE tournament_id = p_tournament_id
    AND status IN ('pending', 'approved');

  IF v_filled_slots >= v_capacity THEN
    RETURN json_build_object('success', false, 'error', 'Tournament is full');
  END IF;

  -- Insert registration
  INSERT INTO registrations (
    tournament_id, team_name, leader_name, leader_game_id,
    leader_whatsapp, transaction_id, payment_screenshot_url
  ) VALUES (
    p_tournament_id, p_team_name, p_leader_name, p_leader_game_id,
    p_leader_whatsapp, p_transaction_id, p_payment_screenshot_url
  ) RETURNING id INTO v_registration_id;

  -- Insert participants if any
```

```
  IF p_participants IS NOT NULL THEN
    INSERT INTO participants (registration_id, player_name, player_game_id,
slot_position)
    SELECT v_registration_id, item->>'name', item->>'game_id',
(item->>'position')::INTEGER
    FROM jsonb_array_elements(p_participants) AS item;
  END IF;

  RETURN json_build_object(
    'success', true,
    'registration_id', v_registration_id,
    'filled_slots', v_filled_slots + 1,
    'remaining_slots', v_capacity - v_filled_slots - 1
  );
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;
get_slot_availability - Real-time slot checking


CREATE OR REPLACE FUNCTION get_slot_availability(p_tournament_id UUID)
RETURNS JSON AS $$
DECLARE
  v_capacity INTEGER;
  v_filled INTEGER;
BEGIN
  SELECT t.max_capacity, COUNT(r.id)
  INTO v_capacity, v_filled
  FROM tournaments t
  LEFT JOIN registrations r ON r.tournament_id = t.id
    AND r.status IN ('pending', 'approved')
  WHERE t.id = p_tournament_id
  GROUP BY t.max_capacity;

  RETURN json_build_object(
    'capacity', v_capacity,
    'filled', v_filled,
    'remaining', v_capacity - v_filled
  );
END;
$$ LANGUAGE plpgsql;
Row Level Security (RLS)
tournaments - Public read access


ALTER TABLE tournaments ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Anyone can view tournaments"
  ON tournaments FOR SELECT
  TO public
  USING (true);
```

registrations - Restricted access

```
ALTER TABLE registrations ENABLE ROW LEVEL SECURITY;

-- Anonymous users can insert via RPC only (handled by function)
CREATE POLICY "Anyone can insert registrations"
  ON registrations FOR INSERT
  TO public
  WITH CHECK (true);

-- Only admin can read all registrations
CREATE POLICY "Admin can view all registrations"
  ON registrations FOR SELECT
  TO authenticated
  USING (
    EXISTS (
      SELECT 1 FROM auth.users
      WHERE id = auth.uid()
      AND raw_user_meta_data->>'role' = 'admin'
    )
  );

-- Only admin can update status
CREATE POLICY "Admin can update registrations"
  ON registrations FOR UPDATE
  TO authenticated
  USING (
    EXISTS (
      SELECT 1 FROM auth.users
      WHERE id = auth.uid()
      AND raw_user_meta_data->>'role' = 'admin'
    )
  );
```

participants - Follow registration policies

```
ALTER TABLE participants ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Admin can view participants"
  ON participants FOR SELECT
  TO authenticated
  USING (
    EXISTS (
      SELECT 1 FROM auth.users
      WHERE id = auth.uid()
      AND raw_user_meta_data->>'role' = 'admin'
    )
  );
```

admin_actions - Admin only

```
ALTER TABLE admin_actions ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Admin can manage actions"
  ON admin_actions FOR ALL
  TO authenticated
  USING (
    EXISTS (
      SELECT 1 FROM auth.users
      WHERE id = auth.uid()
      AND raw_user_meta_data->>'role' = 'admin'
    )
  );
```

📝 Validation Schemas (Zod)

Common Validators

```
// lib/validation/common.ts

// Name validation - Only letters and spaces
export const nameSchema = z.string()
  .min(2, "Name must be at least 2 characters")
  .max(50, "Name must be less than 50 characters")
  .regex(/^[A-Za-z\s]+$/, "Name can only contain letters and spaces")
  .trim();

// Indian mobile number - 10 digits starting with 6-9
export const indianMobileSchema = z.string()
  .regex(/^[6-9]\d{9}$/, "Please enter a valid 10-digit Indian mobile number");

// Game ID validation
export const gameIdSchema = z.string()
  .min(3, "Game ID must be at least 3 characters")
  .max(50, "Game ID must be less than 50 characters")
  .trim();

// Transaction ID
export const transactionIdSchema = z.string()
  .min(5, "Transaction ID must be at least 5 characters")
  .trim();

// File validation
export const paymentScreenshotSchema = z
  .instanceof(File)
  .refine((file) => file.size <= 5 * 1024 * 1024, "File size must be less than
5MB")
  .refine(
    (file) => ["image/jpeg", "image/jpg", "image/png"].includes(file.type),
    "Only JPEG and PNG images are allowed"
  );
```

BGMI Forms

```
// lib/validation/bgmi.ts

export const bgmiSoloSchema = z.object({
  leader_name: nameSchema,
  leader_game_id: gameIdSchema,
  leader_whatsapp: indianMobileSchema,
  transaction_id: transactionIdSchema,
  payment_screenshot: paymentScreenshotSchema,
});

export const bgmiDuoSchema = z.object({
  team_name: z.string().min(3).max(50).trim(),
  leader_name: nameSchema,
  leader_game_id: gameIdSchema,
  leader_whatsapp: indianMobileSchema,
  player2_name: nameSchema,
  player2_game_id: gameIdSchema,
  transaction_id: transactionIdSchema,
  payment_screenshot: paymentScreenshotSchema,
});

export const bgmiSquadSchema = z.object({
  team_name: z.string().min(3).max(50).trim(),
  leader_name: nameSchema,
  leader_game_id: gameIdSchema,
  leader_whatsapp: indianMobileSchema,
  player2_name: nameSchema,
  player2_game_id: gameIdSchema,
  player3_name: nameSchema,
  player3_game_id: gameIdSchema,
  player4_name: nameSchema,
  player4_game_id: gameIdSchema,
  transaction_id: transactionIdSchema,
  payment_screenshot: paymentScreenshotSchema,
});
```

🎨 Modern UI Features

1. Homepage

Hero section with video background (auto-play, muted, loop)
Animated statistics counter (total tournaments, active players)
Game introduction sections with images
Call-to-action buttons with hover effects
Smooth scroll animations (Framer Motion)
Particle effects in background

2. Registration Forms

Multi-step progress indicator
Real-time validation feedback
Typing sound effects on input
Smooth field transitions

Image preview before upload
Optimistic UI updates
Success confetti animation
Live slot availability meter with color coding:
Green: > 50% available
Yellow: 20-50% available
Red: < 20% available
Pulsing animation when slots are low
3. Animations (Framer Motion)
Page transitions with fade + slide
Card hover effects with 3D transform
Staggered list animations
Loading skeletons with shimmer
Smooth modal animations
Number count-up animations
4. Typography & Colors
Font: Inter (modern, readable)
Headings: Bold, gradient text effects
Color scheme:
Primary: Blue gradient (#3B82F6 to #8B5CF6)
BGMI: Orange accent (#F59E0B)
Free Fire: Red accent (#EF4444)
Success: Green (#10B981)
Warning: Amber (#F59E0B)
Dark mode support with next-themes
5. Sound Effects
Typing sound on text inputs (toggleable)
Success sound on form submission
Error sound on validation failure
Stored in localStorage for preference
🔄 Real-time Features (Supabase Realtime)
Slot Tracking

```
// lib/hooks/useSlotTracking.ts
export function useSlotTracking(tournamentId: string) {
  const [slots, setSlots] = useState({ filled: 0, capacity: 0, remaining: 0 });

  useEffect(() => {
    // Subscribe to registrations table changes
    const channel = supabase
      .channel(`tournament-${tournamentId}`)
      .on(
        'postgres_changes',
        {
          event: '*', // INSERT, UPDATE, DELETE
          schema: 'public',
          table: 'registrations',
          filter: `tournament_id=eq.${tournamentId}`
        },
        async () => {
```

```
          // Fetch updated slot count
          const { data } = await supabase.rpc('get_slot_availability', {
            p_tournament_id: tournamentId
          });
          setSlots(data);
        }
      )
      .subscribe();

    return () => {
      channel.unsubscribe();
    };
  }, [tournamentId]);

  return slots;
}
```

Admin Panel Real-time Updates

Live updates to registration list when new submissions arrive

Instant status badge updates when admin approves/rejects

Toast notifications for new registrations

No manual refresh needed

🔐 Admin Panel Features

Authentication

Email/password login via Supabase Auth

Admin user created with metadata: { role: 'admin' }

Protected routes with authentication guard

Session persistence

Dashboard Layout

Sidebar navigation

Game/Mode filter tabs (BGMI/Free Fire → Solo/Duo/Squad)

Status filter (All/Pending/Approved/Rejected)

Search functionality (by team name, player name, transaction ID)

Export to CSV button

Registration Cards

Compact card layout with key info

Expandable details section

Payment screenshot with zoom modal

Approve/Reject buttons with confirmation

Reject requires reason input

Color-coded status badges

Timestamp display

Actions

Approve: Updates status to 'approved', sends WhatsApp notification (manual for now)

Reject: Updates status to 'rejected', frees up slot, logs reason

View Details: Modal with all registration data

Zoom Screenshot: Full-screen image modal with pan/zoom

Add Supabase Realtime for live slot tracking and updates
Build 6 separate modern registration forms (BGMI/Free Fire: Solo/Duo/Squad)
Create comprehensive admin panel with approval system and real-time updates
Implement modern UI with animations, typing sounds, and responsive design
Test all features and prepare for Vercel deployment

Configure Supabase client and create database schema with proper tables, RLS policies, and functions
Fix data loss issues on refresh and sharing by implementing proper database persistence

NEXT_PUBLIC_SUPABASE_URL=https://ielwxcdoejxahmdsfznj.supabase.co
NEXT_PUBLIC_SUPABASE_PUBLISHABLE_DEFAULT_KEY=sb_publishable_OOQjSct460-UZ6xSOJrmiw_JgfGPRuQ

GameArena Tournament Platform Environment Variables
Created for Next.js deployment with Supabase integration

# Supabase Configuration
NEXT_PUBLIC_SUPABASE_URL=https://ielwxcdoejxahmdsfznj.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6ImllbHd4Y2RvZWp4YWhtZHNmem5qIiwicm9sZSI6ImFub24iLCJpYXQiOjE3MzUzNzg0NjIsImV4cCI6MjA1MDk1NDQ2Mn0.ZqX7_3QtMVf4W1jdtafWBoOMuy9_DDlABJYK39aRYGo

# Admin Configuration
NEXT_PUBLIC_ADMIN_EMAIL=ishukriitpatna@gmail.com

# Development Environment
NODE_ENV=development

NEXT_SUPABASE_PROJECT_ID="ielwxcdoejxahmdsfznj"

NEXT_ADMIN_EMAIL=ishukriitpatna@gmail.com
NEXT_ADMIN_PASSWORD=ISHUkr75@

VITE_SUPABASE_URL=https://ielwxcdoejxahmdsfznj.supabase.co
NEXT_SUPABASE_PUBLISHABLE_DEFAULT_KEY=sb_publishable_OOQjSct460-UZ6xSOJrmiw_JgfGPRuQ

and mujhe vercel ar deploy karna hai to usme bhe koi error nhi aana chiaye and sab cheez fully working karna chiaye and api key invalid aata hai kabhi kabhi to usko bhe sahi karna hai and website ko aacha se debug karo haar ek files and folder and whole website ko aacha se debug karo agar koi error hoga to usko bhe sahi kar dena pls and sar pages and mera whole website modern hona chiaye and thoda bhout animated bhe hona chiaye and bhout sare libraies ka use karo and sare pages mai bhout sare lirbaies ka use karo pls and suberbase ke liye jitna bhe code likha hoga sab aacha se likh lena

and ek contact page bhe create karo usme pura details add kar dena tum apne according

and contact page mai or bhe jada detailing add kar dena and sare pages mai or bhe jada libraies ka use karo and sab cheez mujhe fully working mai chiaye and sare pages ka data fully logic

superbase ke liye to pura code likho jitna bhe likha jaa sakte and admin panel ke liye bhe

And sare pages mai tum apna logic and mind use karna pls