THE SANDBOX 3.0

GENERAL OVERVIEW & SYSTEM SUMMARY

Important Note: This architecture documentation represents the initial design phase. The actual implementation may evolve based on development requirements, technical constraints, and stakeholder feedback. Always refer to the latest version and consult with the development team for current specifications.

1 — SYSTEM OVERVIEW

What is The Sandbox 3.0?

The Sandbox 3.0 is a comprehensive web-based competition management system designed specifically for IEEE ITB Student Branch to manage their annual technology competition events. The system handles the complete lifecycle of three concurrent competitions from registration to winner announcement.

System Scope

In Scope:

- User authentication (credentials + OAuth)
- Team registration with member management
- Multi-stage submission system (preliminary, semifinal, final)
- Payment processing and verification
- Admin review and approval workflows
- Email notification system
- Document management (file uploads)
- Real-time status tracking

Out of Scope:

- Automated payment gateway (Midtrans/Xendit) - Manual verification used
- Public voting system - No community voting
- Live chat support - Email-based communication
- Exhibition/Grand Seminar ticketing - Separate from competition system
- Score calculation automation - Manual judging by committee

---

2 — BUSINESS REQUIREMENTS

Competition Types

The system manages three distinct competitions simultaneously:

Competition	Code	Team Size	Focus Area	Final Format
ProtoTech Contest	PTC	3-5 members	Hardware/IoT prototypes	Live pitching + demo
Technovate Paper Competition	TPC	1-3 members	Research papers	Live presentation
Business Case Competition	BCC	3 members (fixed)	Business solutions	Pitch deck + live pitch

Key Business Rules

1. One User = One Competition

- Each registered user can only participate in ONE competition
- Cannot register for multiple competitions
- Cannot be member of multiple teams
- Enforced via database UNIQUE constraint

Rationale: Prevent resource splitting and ensure commitment

2. One Email = One Team

- Each email address can only be used ONCE across all teams
- Global uniqueness across all competitions
- Leader email must match their registered account

Rationale: Prevent duplicate registrations and ensure authenticity

3. Team Name Uniqueness

- Team names must be unique globally
- Cannot reuse names from other teams in any competition

Rationale: Clear identification and prevent confusion

4. Leader Authority

- Only the team leader can perform submissions
- Leader is the user who created the registration
- All members receive email notifications

Rationale: Clear responsibility and accountability

5. Sequential Phase Progression

- Teams must complete phases in order: Registration → Preliminary → Payment → Semifinal → Final
- Cannot skip phases
- Each phase has specific deadlines

Rationale: Structured evaluation and fair competition

6. Payment Critical Path

- Payment section ONLY appears after preliminary qualification
- Payment rejection = immediate elimination (no resubmission)
- Must pay before deadline or team is eliminated

Rationale: Financial commitment and serious participation

7. Final Submission Conditional

- BCC only: Finalists must submit pitch deck
- PTC & TPC: Finalists go directly to live pitching (no file submission)

Rationale: Competition-specific requirements

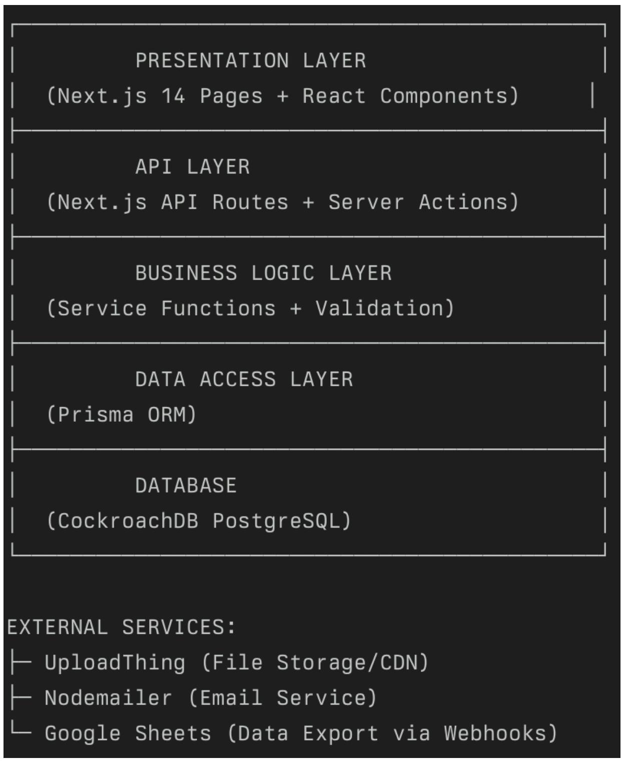
---

3 — SYSTEM ARCHITECTURE SUMMARY

Architecture Pattern

Type: Monolithic Next.js Full-Stack Application
Pattern: Server-Side Rendering (SSR) + API Routes
Database: Single PostgreSQL database (CockroachDB hosted)
Authentication: NextAuth.js with JWT sessions

System Layers



```
┌─────────────────────────────────────────────┐
│         PRESENTATION LAYER                  │
│  (Next.js 14 Pages + React Components)     │
├─────────────────────────────────────────────┤
│         API LAYER                           │
│  (Next.js API Routes + Server Actions)      │
├─────────────────────────────────────────────┤
│         BUSINESS LOGIC LAYER                │
│  (Service Functions + Validation)           │
├─────────────────────────────────────────────┤
│         DATA ACCESS LAYER                   │
│  (Prisma ORM)                               │
├─────────────────────────────────────────────┤
│         DATABASE                            │
│  (CockroachDB PostgreSQL)                   │
└─────────────────────────────────────────────┘


EXTERNAL SERVICES:
├─ UploadThing (File Storage/CDN)
├─ Nodemailer (Email Service)
└─ Google Sheets (Data Export via Webhooks)
```

Component Breakdown

Layer	**Technology**	**Purpose**
Frontend	Next.js 14, React 18, TailwindCSS	User interface, forms, dashboards
Backend	Next.js API Routes, Server Actions	REST API endpoints, server logic
Database	Prisma ORM + CockroachDB	Data persistence, relational data
Authentication	NextAuth.js	User login, sessions, OAuth

File Storage	UploadThing	PDF/image uploads, CDN delivery
Email	Nodemailer	Transactional emails
Validation	Zod	Runtime type checking, form validation

---

4 — TECH STACK

Core Technologies

Frontend:

- Framework: Next.js 14 (App Router or Pages Router)
- UI Library: React 18
- Styling: TailwindCSS 3
- State Management: React Context / Zustand
- Form Handling: React Hook Form + Zod validation
- UI Components: Shadcn/ui (optional)

Backend:

- Runtime: Node.js 18+
- Framework: Next.js API Routes
- ORM: Prisma 5
- Authentication: NextAuth.js 4
- Validation: Zod
- Email: Nodemailer

Database:

- Type: PostgreSQL (via CockroachDB)
- ORM: Prisma
- Migrations: Prisma Migrate
- Seeding: Prisma Seed Scripts

External Services:

- File Storage: UploadThing (https://uploadthing.com)
- Email Provider: SMTP (Gmail/SendGrid/AWS SES)
- Webhooks: Google Apps Script (Sheets integration)
- OAuth Providers: Google (future: GitHub, Facebook)

Development Tools:

- Version Control: Git + GitHub
- Package Manager: npm / pnpm / yarn
- Code Quality: ESLint + Prettier
- Type Safety: TypeScript 5
- Testing: Jest + React Testing Library (optional)

Environment Variables

```
# Database
DATABASE_URL="postgresql://..."

# NextAuth
NEXTAUTH_URL="http://localhost:3000"
NEXTAUTH_SECRET="random-secret-string"

# OAuth Providers
GOOGLE_CLIENT_ID="..."
GOOGLE_CLIENT_SECRET="..."

# UploadThing
UPLOADTHING_SECRET="..."
UPLOADTHING_APP_ID="..."

# Email
SMTP_HOST="smtp.gmail.com"
SMTP_PORT="587"
SMTP_USER="noreply@sandbox.com"
SMTP_PASS="..."
EMAIL_FROM="The Sandbox <noreply@sandbox.com>"

# Google Sheets Webhooks
SHEET_WEBHOOK_REGISTRATION="https://script.google.com/..."
SHEET_WEBHOOK_PRELIMINARY="https://script.google.com/..."
SHEET_WEBHOOK_PAYMENT="https://script.google.com/..."
SHEET_WEBHOOK_SEMIFINAL="https://script.google.com/..."
SHEET_WEBHOOK_FINAL="https://script.google.com/..."

# App Config
```

```
NODE_ENV="development"
```

---

5 — CORE FEATURES

For Participants (Users)

1. Account Management

- ✅ Register account with email verification
- ✅ Login via credentials or Google OAuth
- ✅ Password reset functionality
- ✅ Profile management

2. Team Registration

- ✅ Browse available competitions (PTC, TPC, BCC)
- ✅ Create team with unique name
- ✅ Add team members (3-5 for PTC, 1-3 for TPC, 3 for BCC)
- ✅ Upload proof documents (twibbon, KTM, posters) via Google Drive
- ✅ Real-time validation (team name uniqueness, email uniqueness)
- ✅ Submit for admin approval

3. Preliminary Submission

- ✅ Upload abstract/proposal (PDF, max 10MB)
- ✅ Deadline enforcement
- ✅ View submission status (pending/qualified/rejected)
- ✅ Receive email notifications

4. Payment Submission

- ✅ View payment details (amount, bank account)
- ✅ Upload payment proof (image, max 5MB)
- ✅ Fill payer information
- ✅ Track verification status
- ✅ Critical: Rejection = elimination

5. Semifinal Submission

- ✅ Competition-specific forms:
  - PTC: Upload full paper + YouTube video link
  - TPC: Upload full proposal
  - BCC: Upload full case proposal
- ✅ Deadline tracking
- ✅ Submission confirmation

6. Final Phase (BCC Only)

- ✅ View finalist status
- ✅ Upload pitch deck (PDF/PPT, max 20MB)
- ✅ Grand final event information

7. Dashboard & Tracking

- ✅ Real-time status display
- ✅ Current phase indicator
- ✅ Next action prompts
- ✅ Deadline countdowns
- ✅ Team information view
- ✅ Submission history

For Administrators

1. Registration Management

- ✅ View pending registrations
- ✅ Review team details and member documents
- ✅ Approve or reject registrations with notes
- ✅ Bulk actions (optional)
- ✅ Filter by competition type

2. Preliminary Review

- ✅ View submitted abstracts/proposals
- ✅ Download files for review
- ✅ Qualify or reject submissions
- ✅ Add review notes/feedback
- ✅ Track review progress

3. Payment Verification

- ✅ View pending payments (Finance/Super Admin only)
- ✅ Review payment proof images
- ✅ Verify or reject payments
- ✅ Match amounts with bank statements
- ✅ Add rejection reasons

4. Semifinal Evaluation

- ✅ Access submitted semifinal works
- ✅ Download files (papers, proposals)
- ✅ View video links (PTC)
- ✅ Input evaluation scores (optional feature)

- ✅ Select finalists (bulk selection)

5. Finalist Management

- ✅ Announce finalists (sends emails automatically)
- ✅ View final submissions (BCC)
- ✅ Access pitch decks
- ✅ Export finalist data

6. Reporting & Analytics

- ✅ View registration statistics
- ✅ Export data to Google Sheets (auto-sync)
- ✅ Track submission rates per phase
- ✅ Monitor deadline compliance
- ✅ Generate summary reports

7. User Management

- ✅ View all registered users (Super Admin only)
- ✅ Deactivate accounts if needed
- ✅ Reset user passwords
- ✅ Manage admin accounts

---

6 — USER ROLES & PERMISSIONS

User Roles

Role	**Description**	**Access Level**
Participant (User)	Registered team members	Read own data, submit for own team
Super Admin	Full system access	All features, all data

Moderator	Competition management	Registration & submission review only
Finance	Payment verification	Payment section only

Permission Matrix

Feature	**Participant**	**Moderator**	**Finance**	**Super Admin**
Register account	✅	❌	❌	✅
Create team	✅	❌	❌	✅
Submit works	✅ (own)	❌	❌	✅
View own data	✅	❌	❌	✅
Approve registrations	❌	✅	❌	✅
Review submissions	❌	✅	❌	✅
Verify payments	❌	❌	✅	✅

Select finalists	❌	✅	❌	✅
Manage admins	❌	❌	❌	✅
View reports	❌	✅	✅	✅
Export data	❌	✅	❌	✅

Access Control Implementation

typescript
```
// Middleware example
export function requireAuth(roles?: AdminRole[]) {
  return async (req, res) => {
    const session = await getServerSession(req, res, authOptions);

    if (!session) {
      return res.status(401).json({ error: 'Unauthorized' });
    }

    if (roles && !roles.includes(session.admin.adminRole)) {
      return res.status(403).json({ error: 'Forbidden' });
    }

    return { session };
  };
}

// Usage in API route
export default async function handler(req, res) {
  const { session } = await requireAuth(['super_admin', 'finance'])(req, res);

  // Payment verification logic
}
```

7 — COMPETITION FLOW SUMMARY

Complete Journey Map

```
START: User visits website
│
├─ [PHASE 0] AUTHENTICATION
│  ├─ Register account → Email verification → Activated
│  └─ Login with credentials or Google OAuth
│
├─ [PHASE 1] TEAM REGISTRATION
│  ├─ Browse competitions (PTC/TPC/BCC)
│  ├─ Fill team form (name, institution, members)
│  ├─ Submit → Status: Pending
│  └─ WAIT: Admin approval
│
├─ [DECISION POINT 1] Admin Reviews Registration
│  ├─ ✅ APPROVED → Unlock Preliminary
│  └─ ❌ REJECTED → END (cannot continue)
│
├─ [PHASE 2] PRELIMINARY SUBMISSION
│  ├─ Upload abstract/proposal (PDF)
│  ├─ Submit → Status: Pending
│  └─ WAIT: Admin review
│
├─ [DECISION POINT 2] Admin Reviews Abstract
│  ├─ ✅ QUALIFIED → Unlock Payment
│  └─ ❌ REJECTED → END (cannot continue)
│
├─ [PHASE 3] PAYMENT
│  ├─ View payment details
│  ├─ Transfer to bank account
│  ├─ Upload payment proof
│  ├─ Submit → Status: Pending
│  └─ WAIT: Finance verification
│
├─ [DECISION POINT 3] Finance Verifies Payment
│  ├─ ✅ VERIFIED → Unlock Semifinal
│  └─ ❌ REJECTED → END (ELIMINATED - no second chance)
│
├─ [PHASE 4] SEMIFINAL SUBMISSION
│  ├─ PTC: Upload full paper + YouTube video link
│  ├─ TPC: Upload full proposal
│  ├─ BCC: Upload case proposal
│  ├─ Submit → Status: Submitted
│   └─ WAIT: Admin evaluation + finalist selection
│
├─ [DECISION POINT 4] Admin Selects Finalists
│   ├─ ✅ SELECTED AS FINALIST → Proceed to Final
│   └─ ❌ NOT SELECTED → END (thanked for participation)
│
├─ [PHASE 5] FINAL PHASE
│   ├─ PTC: Live pitching at grand final (no submission)
│   ├─ TPC: Live presentation at grand final (no submission)
│   └─ BCC: Submit pitch deck → Live pitching at grand final
│
└─ [PHASE 6] GRAND FINAL EVENT
   ├─ Live pitching/presentation
   ├─ Judging by panel
   └─ Winner announcement (Juara 1, 2, 3)

END: Competition complete
```

---

8 — DATA MODELS OVERVIEW

Entity Relationship Summary

The system consists of 14 core models organized into 3 categories:

Category 1: Authentication (5 models)

- User - Participant accounts
- Account - OAuth provider data
- Session - Login sessions
- ActivateToken - Email verification tokens
- ResetToken - Password reset tokens

Category 2: Competition System (8 models)

- Competition - Competition configurations (PTC, TPC, BCC)
- CompetitionRegistration - User participation record
- Team - Team information
- TeamMember - Team member details
- PreliminarySubmission - Abstract/proposal submissions
- Payment - Payment records
- SemifinalSubmission - Semifinal work submissions
- FinalSubmission - Final pitch deck (BCC only)

Category 3: Administration (1 model)

- Admin - Admin accounts with roles

Key Relationships

```
User (1) ←→ (1) CompetitionRegistration ←→ (1) Team
                      ↓
                      ├─ (0..1) PreliminarySubmission
                      ├─ (0..1) Payment
                      ├─ (0..1) SemifinalSubmission
                      └─ (0..1) FinalSubmission

Team (1) ←→ (N) TeamMember

Competition (1) ←→ (N) CompetitionRegistration

Admin (1) ←→ (N) PreliminarySubmission (reviewer)
Admin (1) ←→ (N) Payment (verifier)
```

Critical Constraints

Constraint	Model	Field	Purpose
UNIQUE	CompetitionRegistration	userId	One user = one competition
UNIQUE	TeamMember	email	One email = one team globally
UNIQUE	Team	teamName	Team names must be unique
UNIQUE	PreliminarySubmission	registrationId	One submission per registration

UNIQUE	Payment	registrationId	One payment per registration
UNIQUE	SemifinalSubmission	registrationId	One semifinal work per team
UNIQUE	FinalSubmission	registrationId	One final work per team

Data Flow Example

```
1. User registers → User record created (active: false)
2. User clicks email link → User.active = true
3. User creates team → CompetitionRegistration + Team + TeamMembers created
4. Admin approves → CompetitionRegistration.verificationStatus = 'approved'
5. User submits abstract → PreliminarySubmission created
6. Admin qualifies → PreliminarySubmission.status = 'qualified'
                   → CompetitionRegistration.isPreliminaryQualified = true
7. User submits payment → Payment created
8. Finance verifies → Payment.status = 'verified'
9. User submits semifinal → SemifinalSubmission created
10. Admin selects finalist → CompetitionRegistration.isSemifinalQualified = true
11. User submits final (BCC) → FinalSubmission created
```

---

9 — INTEGRATION POINTS

External Services

1. UploadThing (File Storage)

Purpose: Store and serve uploaded files (PDFs, images)

Integration:

typescript
```typescript
import { createUploadthing } from "uploadthing/next";
```

```typescript
const f = createUploadthing();

export const ourFileRouter = {
  pdfUploader: f({ pdf: { maxFileSize: "10MB" } })
    .middleware(async ({ req }) => {
      const session = await getServerSession(authOptions);
      if (!session) throw new Error("Unauthorized");
      return { userId: session.user.id };
    })
    .onUploadComplete(async ({ metadata, file }) => {
      console.log("Upload complete for userId:", metadata.userId);
      console.log("file url", file.url);
      return { url: file.url };
    }),
};
```

Files Handled:

- Abstract/proposal PDFs (max 10MB)
- Payment proof images (max 5MB)
- Semifinal papers/proposals (max 10MB)
- Pitch decks (max 20MB, PDF/PPT)

CDN URLs:

```
https://utfs.io/f/abc123xyz.pdf
```

---

2. Nodemailer (Email Service)

Purpose: Send transactional emails to participants and admins

Integration:

typescript
```typescript
import nodemailer from 'nodemailer';

const transporter = nodemailer.createTransport({
  host: process.env.SMTP_HOST,
  port: process.env.SMTP_PORT,
  auth: {
    user: process.env.SMTP_USER,
    pass: process.env.SMTP_PASS,
  },
});
```

```
export async function sendEmail({ to, subject, html }) {
  await transporter.sendMail({
    from: process.env.EMAIL_FROM,
    to,
    subject,
    html,
  });
}
```

Email Types:

- Account activation
- Registration confirmation
- Approval/rejection notifications
- Submission confirmations
- Payment verification results
- Finalist announcements

Email Templates:
Using React Email (optional) or plain HTML templates

---

3. Google Sheets (Data Export)

Purpose: Auto-sync registration and submission data for easy panitia tracking

Integration:

typescript
```
// Webhook approach
await fetch(process.env.SHEET_WEBHOOK_REGISTRATION, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    timestamp: new Date().toISOString(),
    teamName,
    institution,
    competitionType,
    members: [...],
  }),
});
```

Separate Sheets:

- Registration data

- Preliminary submissions
- Payment records
- Semifinal submissions
- Final submissions

Webhook Setup:
Google Apps Script deployed as web app, receives POST data and appends to sheet.

---

4. NextAuth.js OAuth Providers

Purpose: Allow users to login via Google (future: GitHub, Facebook)

Integration:

typescript
```
import GoogleProvider from "next-auth/providers/google";

export const authOptions = {
  providers: [
    GoogleProvider({
      clientId: process.env.GOOGLE_CLIENT_ID,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET,
    }),
    // CredentialsProvider for email/password
  ],
};
```

OAuth Flow:

1. User clicks "Login with Google"
2. Redirected to Google consent screen
3. Google returns token
4. NextAuth creates/updates User + Account records
5. Session created, user logged in

---

10 — SECURITY & CONSTRAINTS

Security Measures

1. Authentication Security

- ✅ Passwords hashed with bcrypt (10 rounds)
- ✅ JWT tokens for sessions (HTTP-only cookies)

- ✅ Email verification required before login
- ✅ Password reset with time-limited tokens (1 hour expiry)
- ✅ OAuth support (Google) with secure token handling

2. Authorization

- ✅ Role-based access control (RBAC)
- ✅ API route protection with middleware
- ✅ Server-side session validation on every request
- ✅ Leader-only submission enforcement

3. Input Validation

- ✅ Client-side validation with React Hook Form
- ✅ Server-side validation with Zod schemas
- ✅ SQL injection prevention (Prisma parameterized queries)
- ✅ XSS prevention (React auto-escaping)
- ✅ File type and size validation

4. Data Integrity

- ✅ Database constraints (UNIQUE, NOT NULL, FOREIGN KEY)
- ✅ Transaction usage for multi-table operations
- ✅ Cascade delete for related records
- ✅ Atomic operations for critical updates

5. File Upload Security

- ✅ File type whitelist (PDF, JPG, PNG, PPT only)
- ✅ File size limits enforced
- ✅ Malware scanning (via UploadThing)
- ✅ CDN delivery (no direct server access)

6. Rate Limiting

- ⚠️ To be implemented: API rate limiting per user/IP
- ⚠️ To be implemented: Login attempt throttling

Business Constraints

1. Registration Constraints

typescript
```typescript
// One user = one registration
@@unique([userId]) on CompetitionRegistration

// One email = one team membership
@@unique([email]) on TeamMember
```

```
// Team size limits
minTeamSize: 3, maxTeamSize: 5 (PTC)
minTeamSize: 1, maxTeamSize: 3 (TPC)
minTeamSize: 3, maxTeamSize: 3 (BCC)
```

2. Deadline Enforcement

typescript

```typescript
// Server-side deadline checks
if (new Date() >= competition.preliminaryDeadline) {
  throw new Error('Deadline passed');
}

// Client-side UI disable
const isPastDeadline = new Date() >= deadline;
<Button disabled={isPastDeadline}>Submit</Button>
```

3. Phase Progression

typescript

```typescript
// Cannot skip phases
if (registration.currentPhase !== 'preliminary') {
  throw new Error('Complete previous phase first');
}

// Cannot submit if not qualified
if (!registration.isPreliminaryQualified) {
  throw new Error('Not qualified for payment');
}
```

4. Payment Rules

typescript

```typescript
// Amount must match exactly
if (payment.amount !== competition.registrationFee) {
  throw new Error('Invalid amount');
}

// Rejection = permanent elimination
if (payment.status === 'rejected') {
  // No resubmission allowed
  return 'Team eliminated';
}
```

---

11 — DIFFERENCES FROM SANDBOX 2.0

Major Changes

Feature	**Sandbox 2.0**	**Sandbox 3.0**	**Reason**
User Registration Limit	Multiple tickets allowed	ONE competition only	Prevent resource splitting
Team Code	Required for joining	No team codes	Simplified UX, leader creates complete team
Email Uniqueness	Per-team only	Global across all teams	Prevent duplicate registrations
Payment Timing	Before registration or after abstract	ONLY after preliminary qualified	Financial commitment from serious teams
Payment Rejection	Can resubmit	Instant elimination	Enforce payment accuracy

Competition Count	2 (PTC, H4H)	3 (PTC, TPC, BCC)	Added TPC, removed H4H
Video Upload	File upload to server	YouTube link only	Bandwidth efficiency
Voting System	Public voting (Karya model)	No voting	Pure jury evaluation
Configuration	Hardcoded in code	Database (Competition table)	Easy deadline/fee updates
Exhibition Tickets	Same system	Separate system	Different event type
Midtrans Payment	Automated gateway	Manual verification	Simpler for small scale

Architecture Changes

Database Schema:

```
- TicketCompetition → CompetitionRegistration (with UNIQUE userId)
- ParticipantCompetition → TeamMember (with UNIQUE email)
- Abstract → PreliminarySubmission (generalized)
- Regist3Data → Split into Payment + SemifinalSubmission
- PTCSubmissions + H4HSubmissions → Merged into SemifinalSubmission
+ Competition (new config table)
+ FinalSubmission (BCC final phase)
```

- Karya (removed voting)
- TicketExhibition, TicketGS, TransactionDetail (separate events)

API Routes:

```
- /api/ticket/competition → /api/team/register
- /api/regist2 → /api/submission/preliminary
- /api/regist3 → /api/payment/submit + /api/submission/semifinal
+ /api/competition/[code]
+ /api/submission/final
+ /api/admin/* (new admin endpoints)
- /api/voting/* (removed)
- /api/ticket/exhibition (separate system)
```

Session Structure:

```
// v2.0
session.user.ticketsCompetition = [ // Array
  { id, competitionType, verified, stage, teamId }
]

// v3.0
session.user.registration = { // Single object
  id, competitionType, verificationStatus, currentPhase,
  isPreliminaryQualified, isSemifinalQualified
}
```

Migration Considerations

Data Migration:

- Users, Accounts, Sessions, Tokens → No changes needed
- Admin → No changes needed
- TicketCompetition → Needs restructuring to CompetitionRegistration
- Teams → Keep structure, remove teamCode field
- ParticipantCompetition → Add proofOfRegistrationLink field
- Submissions → Needs restructuring to new models

Code Migration:

- Update API endpoints
- Change session callback logic
- Update form validation rules
- Modify dashboard UI for new flow
- Remove voting-related code
- Update email templates