Play-to-Earn Cavaliers Betting App

TECHNICAL FLOW DIAGRAM & ARCHITECTURE REFERENCE

1 System Architecture

🌐 BROWSER (CLIENT)

`GameCard` — Next game display (public)

`BetForm` — Place bet (authenticated)

`BetList` — Bet history (authenticated)

`PointsCard` — Points balance (authenticated)

`AuthButtons` — Sign in/out

`ThemeToggle` / `JerseyPicker`

▼

FETCH() + AUTHORIZATION: BEARER <JWT>

▼

⚡ NEXT.JS FRONTEND (:3000)

`/` — Main page

`/signin` — Login

`/register` — Sign up

`/admin` — Admin panel

`/api/auth/[...nextauth]`

Credentials + Google OAuth

JWT: `{ email, sub, accessToken }`

Encrypted cookie, 30-day expiry

▼

REST API OVER HTTPS

▼

🏗 NESTJS BACKEND (:3001)

Modules

`AuthModule` — login, register, Google

`GamesModule` — odds, settle, cron

`BetsModule` — place & list bets

`HealthModule` — terminus checks

Guards

`AuthGuard` — JWT via jose

`AdminGuard` — x-admin-api-key

Middleware

CORS, ValidationPipe, ThrottlerGuard

AllExceptionsFilter

Resilience

In-memory cache (5 min TTL)

axios-retry (3×, exp backoff)

Circuit breaker (30s cooldown)

Rate limit: 30/min, 5/min POST /bets

▼

🗄️ MONGODB

`users` — email, points, passwordHash

`games` — gameId, teams, spread, scores

`bets` — userId, gameId, selection, status

🌍 THE ODDS API V4

`/odds` — NBA spreads (Cavaliers)

`/scores` — Completed game scores

Free tier: 500 req/month

○ Browser ○ Frontend ○ Backend ○ Database ○ External API

2 API Endpoints

METHOD	PATH	AUTH	DESCRIPTION
GET	/games/next	Public	Next upcoming Cavaliers game (cached 5 min)
POST	/games/next	Admin	Fetch & store game from Odds API
POST	/games/:gameId/settle	Admin	Submit scores, settle all pending bets
POST	/bets	JWT	Place a bet (5 req/min rate limit)
GET	/bets	JWT	List user's bets with game details
POST	/auth/register	—	Create account with email/password
POST	/auth/login	—	Validate credentials, return user
POST	/auth/google	—	Find or create user via Google OAuth
GET	/auth/me	JWT	Get user email + points balance
GET	/health	Public	MongoDB + Odds API health check

3 Database Schemas

users

_id	PK
email	UNIQUE
passwordHash?	string
authProviders[]	string[]
points	number (default 0)
createdAt	auto timestamp
updatedAt	auto timestamp

games

_id	PK
gameId	UNIQUE
homeTeam	string
awayTeam	string
startTime	Date
spread	number (Cavs-relative)
status	upcoming | finished
finalHomeScore?	number
finalAwayScore?	number
createdAt	auto timestamp
updatedAt	auto timestamp

bets

_id	PK
userId	→ USERS
gameId	→ GAMES
selection	cavaliers | opponent
status	pending | won | lost | push
createdAt	auto timestamp
updatedAt	auto timestamp
compound unique index	(USERID, GAMEID)

4 Flow: Placing a Bet

1 **User clicks "Cavaliers" or "Opponent"**
BetForm triggers `handleBet(selection)`

2 **Frontend sends POST /bets**
`Authorization: Bearer <JWT>` + body: `{ gameId, selection }`

3 **AuthGuard validates JWT**
`jose.jwtVerify(token, NEXTAUTH_SECRET)` → extract email → findOrCreate user → `req.user = { userId, email }`

4 **BetsService validates bet**
Check game exists, status is `"upcoming"`, startTime > now

5 **Insert bet into MongoDB**
Compound unique index `(userId, gameId)` prevents duplicates → `409 Conflict` on dupe

6 **Toast notification + refresh BetList**
`201 { bet doc }` → "Bet placed!" toast → `refreshKey++` triggers re-fetch

5 Flow: Settlement

1 **Admin submits scores**
`POST /games/:gameId/settle` with `x-admin-api-key` + `{ finalHomeScore, finalAwayScore }`

2 **Update game → "finished"**
Save scores, set `status: "finished"`, invalidate cache

3

Calculate adjusted margin

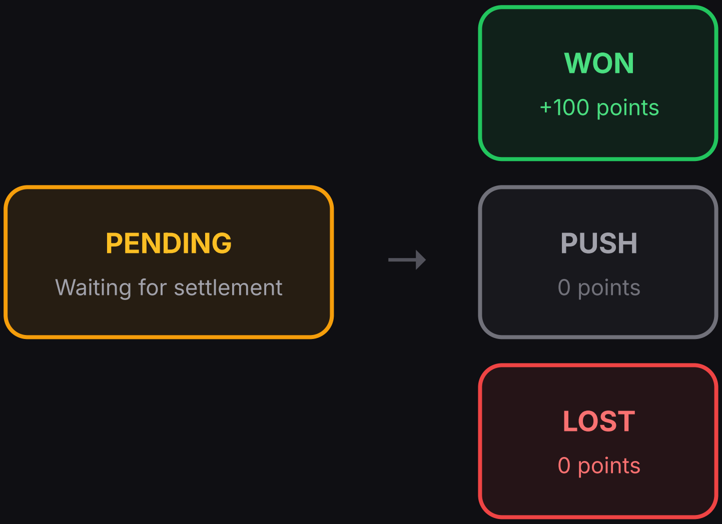
`cavsMargin = (Cavs score - Opponent score)` `adjustedMargin = cavsMargin + spread`

4

Settle each pending bet

`adjustedMargin === 0` → push | `> 0` → cavaliers win | `< 0` → opponent wins | Winners get `+100 points`

6 Bet Lifecycle State Machine



7 Cron Jobs (Every 3 Hours)

REFRESHODDS()

1. `OddsService.fetchNextCavsGame()`
2. GET Odds API → parse Cavaliers spread
3. `GamesService.upsertGame()`
4. Invalidate cache

~160 req/month (within 500 free tier)

AUTOSETTLE()

1. `GamesService.findUnsettledGames()`
2. `OddsService.fetchCompletedScores()`
3. Match gameIds against Scores API
4. `SettleService.settle()` per match

Same settlement logic as admin flow

8 Circuit Breaker & Resilience

OK

CLOSED (Normal)
API calls go through. axios-retry handles transient failures (3 retries, exponential backoff).

FAIL

OPEN (Tripped)
All retries exhausted → circuit opens. Returns stale game from MongoDB as fallback. Cooldown: 30 seconds.

½

HALF-OPEN (Probe)
After 30s, next request probes the API. Success → CLOSED. Failure → OPEN again (reset 30s).

9 Guard Logic

AUTHGUARD (JWT)

1. Extract `Authorization` header
2. Strip `"Bearer "` prefix
3. `jose.jwtVerify(token, NEXTAUTH_SECRET)`
4. Extract email from payload
5. `AuthService.findOrCreateUser(email)`
6. `req.user = { userId, email }`

Missing/invalid → 401 Unauthorized

ADMINGUARD (API KEY)

1. Extract `x-admin-api-key` header
2. Compare with `ADMIN_API_KEY` env var
3. Match → proceed

Missing/wrong → 401 Unauthorized