Game Hub - Architecture & Technical Documentation

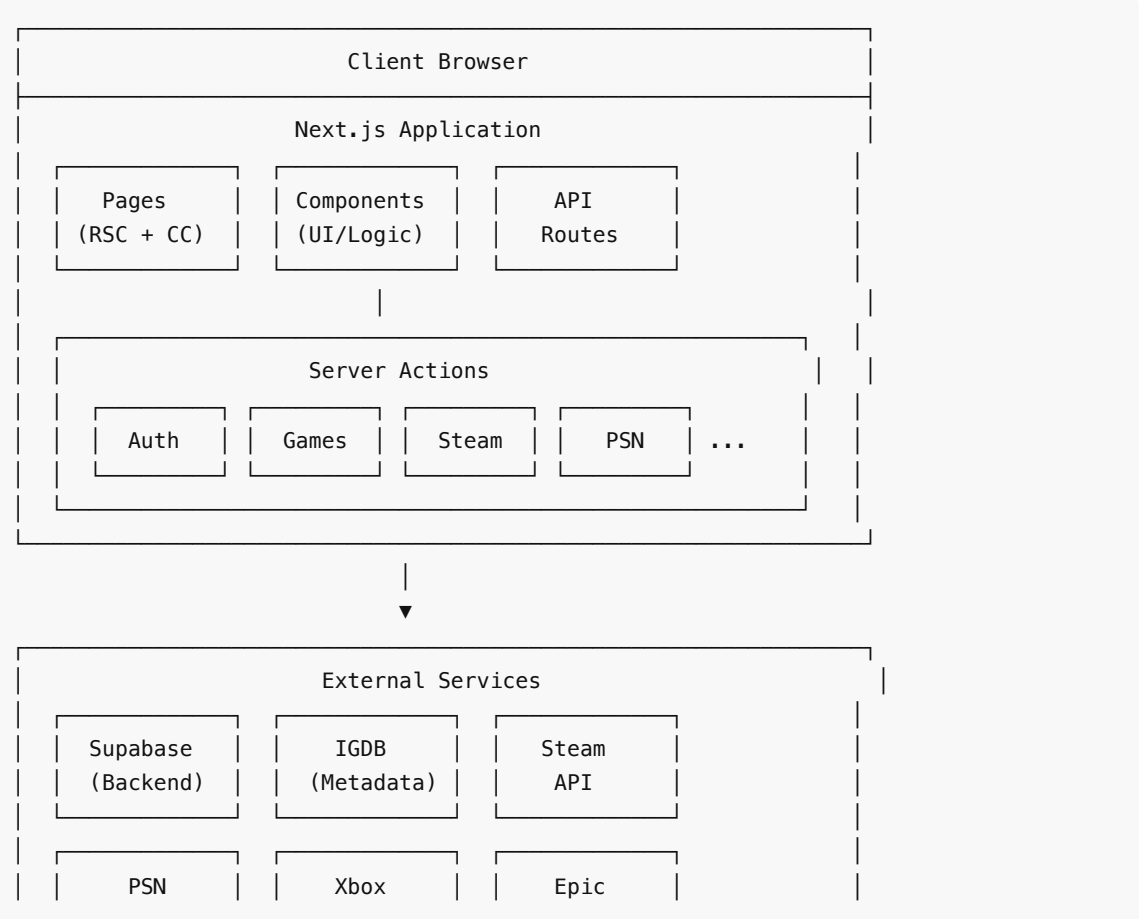
Version: 1.0.0 **Last Updated:** December 27, 2025

Table of Contents

System Architecture

Overview

Game Hub follows a modern Next.js 16 architecture using the App Router pattern with React Server Components as the default.



```
│  │    API      │  │   (OpenXBL) │  │    API      │            │
│  └─────────────┘  └─────────────┘  └─────────────┘            │
└──────────────────────────────────────────────────────────────┘
```

Directory Structure

```
src/
├── app/                          # Next.js App Router
│   ├── (auth)/                   # Auth routes (login, signup, logout)
│   │   ├── error.tsx
│   │   ├── layout.tsx
│   │   ├── login/
│   │   ├── logout/
│   │   └── signup/
│   ├── (dashboard)/              # Protected dashboard routes
│   │   ├── achievements/
│   │   ├── backlog/
│   │   ├── dashboard/
│   │   ├── friends/
│   │   ├── game/[id]/
│   │   ├── library/
│   │   ├── settings/
│   │   └── stats/
│   ├── (marketing)/              # Public marketing pages
│   │   ├── features/
│   │   ├── privacy/
│   │   └── terms/
│   ├── api/                      # API routes
│   │   ├── auth/                 # OAuth callbacks
│   │   ├── games/
│   │   └── igdb/
│   ├── globals.css               # Design system & Tailwind config
│   └── layout.tsx                # Root layout
│
├── components/                   # React components
│   ├── achievements/             # Achievement display components
│   ├── compare/                  # Friend comparison components
│   ├── dashboard/                # Dashboard-specific components
│   │   ├── cards/               # Game cards, stat cards
│   │   └── ...
│   ├── game/                     # Game detail components
│   ├── icons/                    # Platform logos, icons
│   ├── layouts/                  # Shared layouts
│   ├── library/                  # Library page components
│   ├── modals/                   # Modal dialogs
│   │   └── game-form/           # Game form sections
│   ├── settings/                 # Settings page components
│   ├── stats/                    # Analytics components
│   ├── theme/                    # Theme provider, toggles
│   └── ui/                       # shadcn/ui base components
│
```

```
├── lib/                          # Shared utilities & logic
│   ├── actions/                  # Server actions
│   │   ├── achievements.ts
│   │   ├── auth/
│   │   ├── compare/
│   │   ├── epic.ts
│   │   ├── games/
│   │   ├── psn/
│   │   ├── sessions.ts
│   │   ├── stats.ts
│   │   ├── steam/
│   │   └── xbox/
│   ├── constants/                # Shared constants
│   │   ├── index.ts
│   │   └── platforms.ts
│   ├── epic/                     # Epic Games client
│   ├── events/                   # Event system
│   ├── hooks/                    # Custom React hooks
│   ├── igdb/                     # IGDB API client
│   ├── psn/                      # PSN API client
│   ├── steam/                    # Steam API client
│   ├── supabase/                 # Supabase client setup
│   ├── types/                    # TypeScript definitions
│   ├── utils.ts                  # Utility functions
│   └── xbox/                     # Xbox API client
│
└── proxy.ts                      # IGDB API proxy
```

Key Architectural Decisions

1. Server Components First

- Default to React Server Components (RSC)
- Use `"use client"` only when necessary
- Minimize client-side JavaScript bundle

2. Server Actions for Data Mutations

- All data mutations via Server Actions
- Type-safe with TypeScript
- Built-in form handling support

3. Route Groups

- `(auth)` - Authentication flows
- `(dashboard)` - Protected pages with shared layout
- `(marketing)` - Public marketing pages

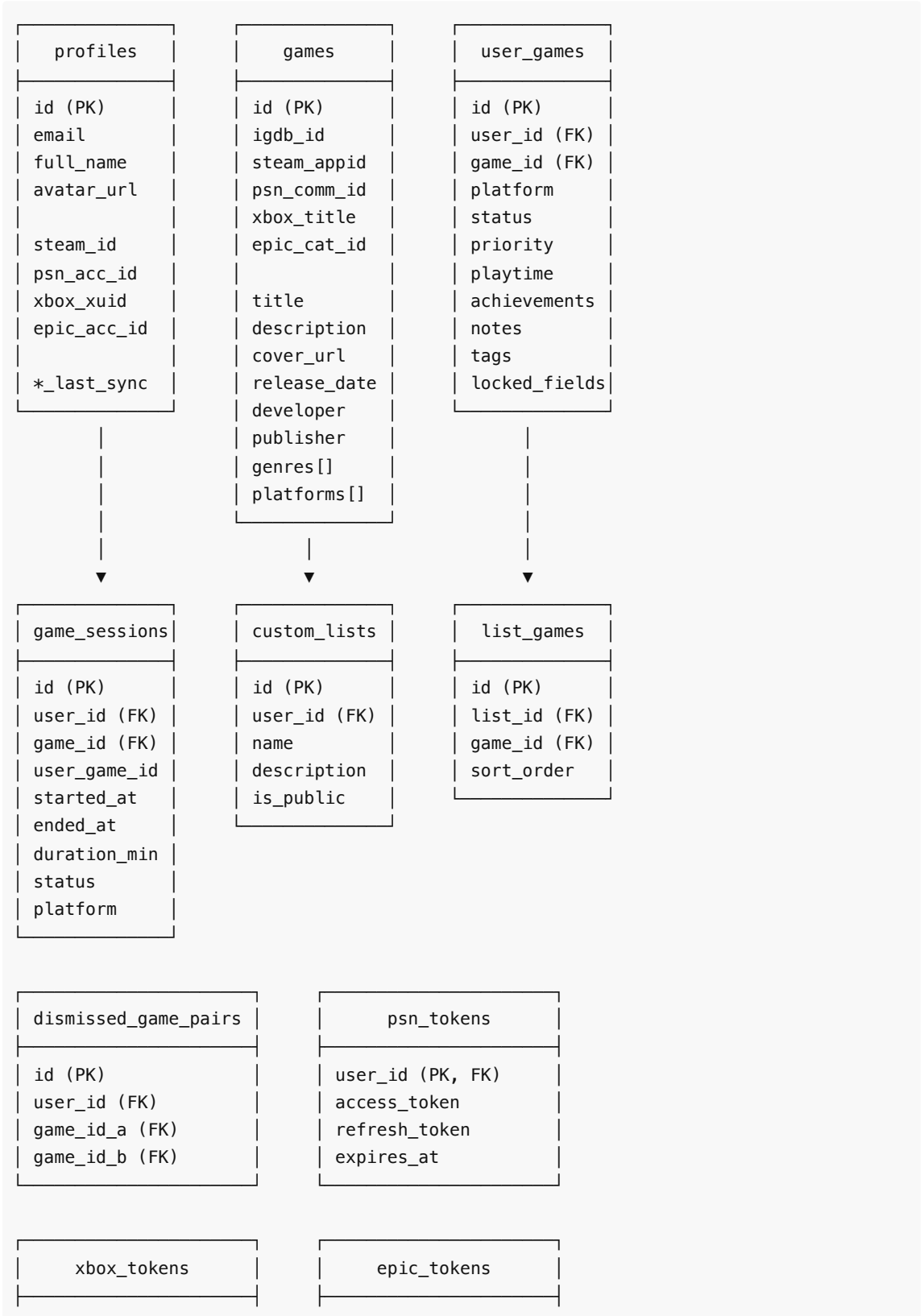
4. API Client Abstraction

Each platform has a dedicated client in `/lib/`:

- Handles authentication
- Manages rate limiting
- Provides type-safe responses

Database Schema

Entity Relationship Diagram



```
| user_id (PK, FK)     |      | id (PK)               |
| api_key              |      | user_id (FK)          |
└──────────────────────┘      | access_token          |
                              | refresh_token         |
                              | expires_at            |
                              | refresh_expires_at    |
                              └───────────────────────┘
```

Table Definitions

profiles

Extended user information beyond Supabase auth.

```sql
CREATE TABLE profiles (
  id UUID REFERENCES auth.users(id) PRIMARY KEY,
  email TEXT UNIQUE NOT NULL,
  full_name TEXT,
  avatar_url TEXT,

  -- Platform connections
  steam_id TEXT UNIQUE,
  steam_persona_name TEXT,
  steam_avatar_url TEXT,
  steam_last_sync TIMESTAMPTZ,

  psn_account_id TEXT UNIQUE,
  psn_online_id TEXT,
  psn_avatar_url TEXT,
  psn_trophy_level INTEGER,
  psn_last_sync TIMESTAMPTZ,

  xbox_xuid TEXT UNIQUE,
  xbox_gamertag TEXT,
  xbox_avatar_url TEXT,
  xbox_gamerscore INTEGER,
  xbox_last_sync TIMESTAMPTZ,

  epic_account_id TEXT UNIQUE,
  epic_display_name TEXT,
  epic_last_sync TIMESTAMPTZ,

  created_at TIMESTAMPTZ DEFAULT now(),
  updated_at TIMESTAMPTZ DEFAULT now()
);
```

games

Master list of all games with metadata.

```sql
CREATE TABLE games (
  id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
```

```
  -- External IDs
  igdb_id INTEGER UNIQUE,
  steam_appid INTEGER UNIQUE,
  psn_communication_id TEXT UNIQUE,
  xbox_title_id TEXT UNIQUE,
  epic_catalog_item_id TEXT UNIQUE,
  epic_namespace TEXT,

  -- Metadata
  title TEXT NOT NULL,
  description TEXT,
  cover_url TEXT,
  release_date DATE,
  developer TEXT,
  publisher TEXT,
  genres TEXT[],
  platforms TEXT[],

  created_at TIMESTAMPTZ DEFAULT now(),
  updated_at TIMESTAMPTZ DEFAULT now()
);
```

user_games

Junction table with user-specific game data.

```
CREATE TABLE user_games (
  id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
  user_id UUID REFERENCES profiles(id) NOT NULL,
  game_id UUID REFERENCES games(id) NOT NULL,

  -- Ownership
  platform TEXT NOT NULL,
  ownership_status TEXT DEFAULT 'owned',
  is_physical BOOLEAN DEFAULT false,
  hidden BOOLEAN DEFAULT false,

  -- Progress
  status TEXT DEFAULT 'unplayed',
  priority TEXT DEFAULT 'medium',
  completion_percentage INTEGER DEFAULT 0,
  playtime_hours DECIMAL(10, 2) DEFAULT 0,
  last_played_at TIMESTAMPTZ,

  -- Personal Data
  personal_rating INTEGER CHECK (1 <= personal_rating <= 10),
  notes TEXT,
  tags TEXT[],
  locked_fields JSONB DEFAULT '{}',

  -- Achievements
```

```
  achievements_earned INTEGER DEFAULT 0,
  achievements_total INTEGER DEFAULT 0,

  -- Platform-specific
  steam_appid INTEGER,
  steam_playtime_minutes INTEGER DEFAULT 0,
  xbox_title_id TEXT,
  psn_title_id TEXT,

  -- Timestamps
  created_at TIMESTAMPTZ DEFAULT now(),
  updated_at TIMESTAMPTZ DEFAULT now(),
  completed_at TIMESTAMPTZ,

  UNIQUE(user_id, game_id, platform)
);
```

game_sessions

Tracks individual gaming sessions.

```
CREATE TABLE game_sessions (
  id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
  user_id UUID REFERENCES profiles(id) NOT NULL,
  game_id UUID REFERENCES games(id) NOT NULL,
  user_game_id UUID REFERENCES user_games(id) NOT NULL,

  started_at TIMESTAMPTZ DEFAULT now(),
  ended_at TIMESTAMPTZ,
  duration_minutes INTEGER,

  status TEXT DEFAULT 'active' CHECK (status IN ('active', 'completed')),
  platform TEXT DEFAULT 'Steam',
  steam_appid INTEGER,

  created_at TIMESTAMPTZ DEFAULT now(),
  updated_at TIMESTAMPTZ DEFAULT now()
);
```

Row Level Security (RLS)

All tables have RLS enabled with policies:

```
-- Users can only access their own data
CREATE POLICY "Users can view their own profile"
  ON profiles FOR SELECT
  USING (auth.uid() = id);

CREATE POLICY "Users can view their own games"
  ON user_games FOR SELECT
  USING (auth.uid() = user_id);
```

```sql
-- Games table is readable by all authenticated users
CREATE POLICY "Anyone can view games"
  ON games FOR SELECT
  USING (true);
```

Database Functions

handle_updated_at

Automatically updates `updated_at` timestamp:

```sql
CREATE FUNCTION handle_updated_at()
RETURNS TRIGGER AS $$
BEGIN
  NEW.updated_at = timezone('utc', now());
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

dismiss_game_pair

Handles duplicate dismissal with normalized ordering:

```sql
CREATE FUNCTION dismiss_game_pair(
  p_user_id UUID,
  p_game_id_1 UUID,
  p_game_id_2 UUID
) RETURNS BOOLEAN AS $$
DECLARE
  v_game_a UUID;
  v_game_b UUID;
BEGIN
  -- Normalize order: smaller UUID first
  IF p_game_id_1 < p_game_id_2 THEN
    v_game_a := p_game_id_1;
    v_game_b := p_game_id_2;
  ELSE
    v_game_a := p_game_id_2;
    v_game_b := p_game_id_1;
  END IF;

  INSERT INTO dismissed_game_pairs (user_id, game_id_a, game_id_b)
  VALUES (p_user_id, v_game_a, v_game_b)
  ON CONFLICT DO NOTHING;

  RETURN TRUE;
END;
$$ LANGUAGE plpgsql;
```

Views

daily_playtime_summary

Aggregates playtime by day:

```sql
CREATE VIEW daily_playtime_summary
WITH (security_invoker = true) AS
SELECT
  user_id,
  DATE(started_at AT TIME ZONE 'UTC') as play_date,
  SUM(COALESCE(duration_minutes, 0)) as total_minutes,
  COUNT(*) as session_count
FROM game_sessions
WHERE status = 'completed'
GROUP BY user_id, DATE(started_at AT TIME ZONE 'UTC');
```

Indexes

Key indexes for performance:

```sql
-- User games lookups
CREATE INDEX idx_user_games_user_id ON user_games(user_id);
CREATE INDEX idx_user_games_status ON user_games(status);
CREATE INDEX idx_user_games_platform ON user_games(platform);
CREATE INDEX idx_user_games_last_played ON user_games(last_played_at DESC);
CREATE INDEX idx_user_games_tags ON user_games USING GIN(tags);

-- Game lookups
CREATE INDEX idx_games_title ON games(title);
CREATE INDEX idx_games_igdb_id ON games(igdb_id);
CREATE INDEX idx_games_steam_appid ON games(steam_appid);
CREATE INDEX idx_games_genres ON games USING GIN(genres);

-- Session lookups
CREATE INDEX idx_game_sessions_user_status ON game_sessions(user_id, status);
CREATE INDEX idx_game_sessions_started_at ON game_sessions(started_at DESC);
```

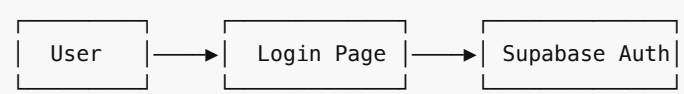
Authentication

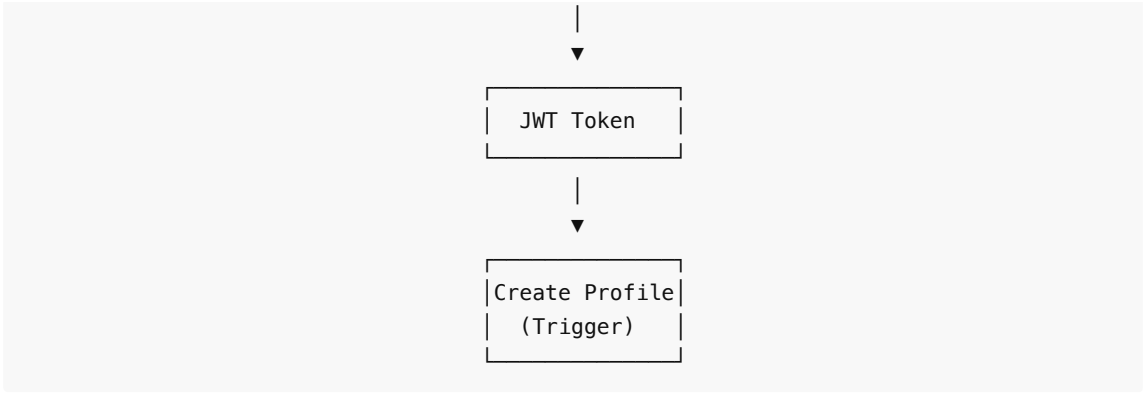
Supabase Auth

Game Hub uses Supabase Auth with:

- Email/password authentication
- OAuth providers (Steam OpenID)
- JWT-based session management

Authentication Flow





Profile Creation Trigger

```sql
CREATE FUNCTION handle_new_user()
RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO profiles (id, email, full_name)
  VALUES (
    NEW.id,
    NEW.email,
    COALESCE(
      NEW.raw_user_meta_data->>'full_name',
      NEW.raw_user_meta_data->>'name',
      split_part(NEW.email, '@', 1)
    )
  );
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER on_auth_user_created
  AFTER INSERT ON auth.users
  FOR EACH ROW EXECUTE FUNCTION handle_new_user();
```

Protected Routes

Dashboard routes are protected via middleware:

```typescript
// src/lib/supabase/server.ts
export async function requireAuth() {
  const supabase = await createClient();
  const { data: { user }, error } = await supabase.auth.getUser();

  if (error || !user) {
    redirect('/login');
  }

  return { user, supabase };
}
```

API Integrations

Steam Web API

Base URL: `https://api.steampowered.com`

Endpoints Used

Endpoint	Method	Description
`/ISteamUser/GetPlayerSummaries/v2/`	GET	Player profile
`/IPlayerService/GetOwnedGames/v1/`	GET	Owned games
`/IPlayerService/GetRecentlyPlayedGames/v1/`	GET	Recent games
`/ISteamUserStats/GetPlayerAchievements/v1/`	GET	Achievements
`/ISteamUserStats/GetSchemaForGame/v2/`	GET	Game schema

Client Implementation

```
// src/lib/steam/client.ts
export async function getOwnedGames(steamId: string): Promise<SteamGame[]> {
  const validSteamId = validateSteamId(steamId);
  const url = `${STEAM_API_BASE}/IPlayerService/GetOwnedGames/v1/?
key=${apiKey}&steamid=${validSteamId}&include_appinfo=1&include_played_free_games=1`;

  const data = await steamApiRequest<GetOwnedGamesResponse>(url);
  return data.response.games || [];
}
```

PlayStation Network API

Library: `psn-api` (npm package)

Functions Used

Function	Description
`exchangeNpssoForCode`	NPSSO to access code
`exchangeCodeForAccessToken`	Code to tokens
`exchangeRefreshTokenForAuthTokens`	Refresh tokens
`getUserTitles`	Trophy titles (library)
`getUserTrophyProfileSummary`	Trophy stats
`getUserTrophiesEarnedForTitle`	Earned trophies
`getTitleTrophies`	Trophy definitions

`getUserPlayedGames`	Playtime data
`makeUniversalSearch`	User search

Client Implementation

```
// src/lib/psn/client.ts
export async function getGameLibrary(
  accessToken: string,
  accountId: string = 'me',
  limit: number = 800
): Promise<PsnTrophyTitle[]> {
  const auth: AuthorizationPayload = { accessToken };
  const response = await getUserTitles(auth, accountId, { limit });

  return response.trophyTitles.map((title) => ({
    npServiceName: title.npServiceName,
    npCommunicationId: title.npCommunicationId,
    trophyTitleName: title.trophyTitleName,
    // ... map other fields
  }));
}
```

Xbox Live API (OpenXBL)

Base URL: `https://xbl.io/api/v2`

Endpoints Used

Endpoint	Description
`/account`	Current user profile
`/account/{xuid}`	Profile by XUID
`/search/{gamertag}`	Search users
`/player/titleHistory`	Game library
`/achievements`	All achievements
`/achievements/player/{xuid}/{titleId}`	Game achievements

IGDB API

Base URL: `https://api.igdb.com/v4`

Authentication

Uses Twitch OAuth with client credentials:

```
async function getAccessToken(): Promise<string> {
  if (tokenCache && tokenCache.expires_at > Date.now() + 3600000) {
```

```
    return tokenCache.access_token;
  }

  const response = await fetch(
    `https://id.twitch.tv/oauth2/token?
client_id=${clientId}&client_secret=${clientSecret}&grant_type=client_credentials`,
    { method: 'POST' }
  );

  const data = await response.json();
  tokenCache = {
    access_token: data.access_token,
    expires_at: Date.now() + 50 * 24 * 60 * 60 * 1000, // 50 days
  };

  return data.access_token;
}
```

Query Language

IGDB uses a custom query language:

```
const query = `
  search "${title}";
  fields name, cover.url, summary, first_release_date,
         release_dates.platform, release_dates.date,
         genres.name, platforms.name,
         involved_companies.company.name,
         involved_companies.developer,
         involved_companies.publisher;
  limit 10;
`;
```

Server Actions

Organization

Server actions are organized by domain:

```
src/lib/actions/
├── achievements.ts      # Achievement statistics
├── auth/                # Authentication actions
├── compare/             # Friend comparison
├── epic.ts              # Epic Games sync
├── game-detail.ts       # Game detail fetching
├── games/
│   ├── crud.ts          # Create, read, update, delete
│   ├── enrichment.ts    # IGDB metadata enrichment
│   ├── index.ts         # Barrel export
│   └── types.ts         # Shared types
├── psn/
```

```
│   ├── auth.ts         # PSN authentication
│   └── sync.ts         # Library sync
├── sessions.ts         # Game session tracking
├── stats.ts            # User statistics
├── steam/
│   ├── auth.ts         # Steam OpenID
│   └── sync.ts         # Library sync
└── xbox/
    ├── auth.ts         # Xbox API key
    └── sync.ts         # Library sync
```

Action Patterns

All server actions follow this pattern:

```typescript
'use server';

import { requireAuth } from '@/lib/supabase/server';
import { revalidatePath } from 'next/cache';

export async function someAction(params: Params) {
  // 1. Authentication
  let user, supabase;
  try {
    ({ user, supabase } = await requireAuth());
  } catch {
    return { error: 'Not authenticated' };
  }

  // 2. Validation (if needed)
  if (!params.isValid) {
    return { error: 'Invalid parameters' };
  }

  try {
    // 3. Business logic
    const result = await doSomething();

    // 4. Revalidate affected paths
    revalidatePath('/dashboard');
    revalidatePath('/library');

    // 5. Return success
    return { success: true, data: result };
  } catch (error) {
    // 6. Error handling
    return {
      error: error instanceof Error ? error.message : 'Unknown error',
    };
  }
}
```

Key Actions

Games CRUD

```
// Create game
export async function addGameToLibrary(gameData: AddGameInput)

// Update game
export async function updateGame(userGameId: string, updates: GameUpdate)

// Delete game
export async function removeGameFromLibrary(userGameId: string)

// Merge duplicates
export async function mergeGames(targetId: string, sourceId: string)
```

Platform Sync

```
// Steam
export async function syncSteamLibrary()
export async function syncSteamAchievements(userGameId: string)

// PSN
export async function authenticatePsn(npsso: string)
export async function syncPsnLibrary()

// Xbox
export async function saveXboxApiKey(apiKey: string)
export async function syncXboxLibrary()

// Epic
export async function exchangeEpicCode(code: string)
export async function syncEpicLibrary()
```

IGDB Enrichment

```
// Enrich all games
export async function enrichAllGamesFromIGDB(platform?: string)

// Update single game cover
export async function updateGameCoverFromIGDB(gameId: string, title: string)

// Refresh release dates
export async function refreshReleaseDatesFromIGDB(platform?: string)
```

Client Components

Custom Hooks

useDashboardData

Fetches dashboard statistics:

```
export function useDashboardData() {
  const [data, setData] = useState<DashboardData | null>(null);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    async function fetchData() {
      const stats = await getDashboardStats();
      setData(stats);
      setIsLoading(false);
    }
    fetchData();
  }, []);

  return { data, isLoading };
}
```

useIGDBSearch

Searches IGDB with debouncing:

```
export function useIGDBSearch(query: string, delay = 300) {
  const [results, setResults] = useState<IGDBGame[]>([]);
  const [isLoading, setIsLoading] = useState(false);

  useEffect(() => {
    if (!query) {
      setResults([]);
      return;
    }

    const timeoutId = setTimeout(async () => {
      setIsLoading(true);
      const searchResults = await searchIGDB(query);
      setResults(searchResults);
      setIsLoading(false);
    }, delay);

    return () => clearTimeout(timeoutId);
  }, [query, delay]);

  return { results, isLoading };
}
```

useSessionTracking

Manages active game sessions:

```tsx
export function useSessionTracking(steamId?: string) {
  const [activeSession, setActiveSession] = useState<GameSession | null>(null);
  const [isChecking, setIsChecking] = useState(false);

  const checkCurrentGame = useCallback(async () => {
    if (!steamId) return;

    setIsChecking(true);
    const currentGame = await getCurrentlyPlayingGame(steamId);

    if (currentGame.isPlaying) {
      // Start or continue session
    } else if (activeSession) {
      // End session
    }

    setIsChecking(false);
  }, [steamId, activeSession]);

  // Poll every 30 seconds
  useEffect(() => {
    const interval = setInterval(checkCurrentGame, 30000);
    return () => clearInterval(interval);
  }, [checkCurrentGame]);

  return { activeSession, isChecking };
}
```

Component Hierarchy

```
DashboardLayout
├── DashboardSidebar
│   └── NavItem[]
├── DashboardHeader
│   └── ThemeToggle
└── [Page Content]
    ├── StatsSection
    │   └── StatCard[]
    ├── NowPlayingSection
    │   └── NowPlayingCard[]
    └── GameLibrary
        └── GameCard[]
```

Type System

Core Types

```tsx
// Game from database
interface Game {
```

```
  id: string;
  igdb_id?: number;
  steam_appid?: number;
  psn_communication_id?: string;
  xbox_title_id?: string;
  epic_catalog_item_id?: string;
  title: string;
  description?: string;
  cover_url?: string;
  release_date?: string;
  developer?: string;
  publisher?: string;
  genres?: string[];
  platforms?: string[];
}

// User's game entry
interface UserGame {
  id: string;
  user_id: string;
  game_id: string;
  game?: Game;
  platform: string;
  status: GameStatus;
  priority: GamePriority;
  completion_percentage: number;
  playtime_hours: number;
  achievements_earned: number;
  achievements_total: number;
  notes?: string;
  tags?: string[];
  // ... more fields
}

type GameStatus = 'unplayed' | 'playing' | 'played' | 'completed' | 'finished' |
'on_hold';
type GamePriority = 'high' | 'medium' | 'low' | 'none' | 'finished';
```

Platform-Specific Types

```
// Steam types
interface SteamGame {
  appid: number;
  name: string;
  playtime_forever: number;
  img_icon_url: string;
  rtime_last_played?: number;
}

// PSN types
interface PsnTrophyTitle {
```

```
  npServiceName: 'trophy' | 'trophy2';
  npCommunicationId: string;
  trophyTitleName: string;
  trophyTitlePlatform: string;
  progress: number;
  earnedTrophies: TrophyCounts;
  definedTrophies: TrophyCounts;
}

// Xbox types
interface XboxTitleHistoryItem {
  titleId: string;
  name: string;
  displayImage: string;
  devices: string[];
  achievement: {
    currentAchievements: number;
    totalAchievements: number;
    currentGamerscore: number;
    totalGamerscore: number;
  };
}
```

Error Types

```
class SteamAPIError extends Error {
  code: string;
  statusCode: number;
}

class SteamPrivacyError extends SteamAPIError {}
class SteamRateLimitError extends SteamAPIError {}
class InvalidSteamIdError extends SteamAPIError {}

// Similar patterns for PSN, Xbox, Epic
```

Design System

Tailwind CSS 4.0 Configuration

Configuration in `globals.css` using `@theme`:

```
@import "tailwindcss";

@theme {
  /* Primary Backgrounds */
  --color-void: #030304;
  --color-abyss: #0a0a0b;
  --color-deep: #0f1011;
```

```
  --color-slate: #161719;
  --color-steel: #1e2023;

  /* Accent Colors */
  --color-cyan-400: #22d3ee;
  --color-cyan-500: #00d9ff;
  --color-violet-400: #a855f7;
  --color-violet-500: #9333ea;
  --color-emerald-400: #34d399;
  --color-amber-400: #fbbf24;

  /* Typography */
  --font-family-display: "Rajdhani", system-ui, sans-serif;
  --font-family-body: "Inter", system-ui, sans-serif;
}
```

CSS Variables for Theming

```
:root {
  /* Light mode */
  --theme-bg-primary: #faf9f7;
  --theme-text-primary: #1a1a1a;
  --theme-accent-cyan: #0891b2;
}

.dark {
  /* Dark mode */
  --theme-bg-primary: #030304;
  --theme-text-primary: #ffffff;
  --theme-accent-cyan: #22d3ee;
}
```

Utility Classes

```
/* Glow effects */
.glow-cyan {
  text-shadow: 0 0 20px rgba(34, 211, 238, 0.5);
}

/* Card styles */
.card-elevated {
  background: var(--theme-bg-secondary);
  border: 1px solid var(--theme-border);
  box-shadow: var(--theme-shadow-lg);
}

/* Button presets */
.btn-primary {
  background-color: var(--theme-text-primary);
```

```
  color: var(--theme-bg-primary);
}
```

Animation Keyframes

```
@keyframes fadeInUp {
  from { opacity: 0; transform: translateY(16px); }
  to { opacity: 1; transform: translateY(0); }
}

@keyframes pulseGlow {
  0%, 100% { box-shadow: 0 0 0 0 rgba(34, 211, 238, 0.4); }
  50% { box-shadow: 0 0 20px 4px rgba(34, 211, 238, 0.2); }
}

@keyframes shimmer {
  0% { background-position: 200% center; }
  100% { background-position: -200% center; }
}
```

Security

Row Level Security

All database tables have RLS enabled:

```
ALTER TABLE user_games ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Users can only access own games"
  ON user_games
  FOR ALL
  USING (auth.uid() = user_id);
```

Input Validation

All user inputs are validated:

```
// Steam ID validation
if (!/^7656119\d{10}$/.test(steamId)) {
  throw new InvalidSteamIdError();
}

// NPSSO validation
if (!/^[a-zA-Z0-9]{60,70}$/.test(npsso)) {
  throw new InvalidNpssoError();
}
```

API Key Security

- Steam API key stored server-side only
- Platform tokens stored in database with RLS
- No secrets exposed to client

SQL Injection Prevention

Using Supabase client with parameterized queries:

```
const { data } = await supabase
  .from('user_games')
  .select('*')
  .eq('user_id', user.id)
  .eq('platform', platform); // Safe parameterization
```

Performance

Caching Strategy

Resource	Cache Duration	Strategy
Steam API	5 minutes	`next: { revalidate: 300 }`
IGDB Token	50 days	In-memory cache
Game Schema	24 hours	`next: { revalidate: 86400 }`
Achievements	No cache	`cache: 'no-store'`

Rate Limiting

Each platform client implements rate limiting:

```
class RateLimiter {
  private requests: number[] = [];
  private windowMs: number;
  private maxRequests: number;

  canMakeRequest(): boolean {
    const now = Date.now();
    this.requests = this.requests.filter(t => now - t < this.windowMs);
    return this.requests.length < this.maxRequests;
  }

  recordRequest(): void {
    this.requests.push(Date.now());
  }
}
```

Database Indexes

Strategic indexes for common queries:

```sql
-- Frequently accessed columns
CREATE INDEX idx_user_games_user_id ON user_games(user_id);
CREATE INDEX idx_user_games_status ON user_games(status);
CREATE INDEX idx_user_games_last_played ON user_games(last_played_at DESC);

-- Array columns (GIN indexes)
CREATE INDEX idx_user_games_tags ON user_games USING GIN(tags);
CREATE INDEX idx_games_genres ON games USING GIN(genres);
```

Bundle Optimization

- Server Components by default
- Dynamic imports for heavy components
- Image optimization with Next.js Image
- Font optimization with next/font

Architecture documentation generated December 27, 2025