# Valorant League Management System

## System Design Document

## Table of Contents



## Introduction

The Valorant League Management System is a full-stack web application designed to facilitate the organization, management, and statistical tracking of community-run Valorant esports leagues and tournaments. The platform bridges the gap between casual competitive play and professional esports by providing comprehensive tools for league administration, team management, match tracking, and performance analytics.

### Project Context

- **Target Audience**: Community-run Valorant leagues, amateur tournaments, collegiate esports
- **Scale**: Multi-league support with multiple concurrent seasons
- **Data Source**: Integration with third-party Valorant match data API
- **Deployment**: Web-based platform accessible via modern browsers

### Technology Stack

- **Frontend**: React with TypeScript
- **Backend**: Node.js, Express, TypeScript
- **Database**: PostgreSQL with Prisma ORM
- **Authentication**: JWT-based session management
- **External Integration**: Third-party Valorant match data API

## Purpose

### Primary Objectives

1. **League Administration**: Enable administrators to create and manage competitive leagues with multiple seasons
2. **Team Management**: Allow players to form teams, recruit members, and register for league competitions
3. **Match Tracking**: Provide systematic match scheduling and comprehensive statistics tracking
4. **Performance Analytics**: Deliver detailed player and team statistics derived from official match data
5. **Community Engagement**: Foster competitive play through organized league structure

### Key Differentiators

- **Community-Focused**: Designed specifically for non-official, community-run tournaments
- **Data-Driven**: Leverages official Valorant match data for accurate statistics
- **Comprehensive Statistics**: Tracks advanced metrics (ADR, ACS, KAST, etc.) beyond basic K/D ratios
- **Role-Based Access**: Distinct workflows for administrators, team captains, and players

---

## Critical Requirements Analysis

### Functional Requirements

#### User Management

- **FR-1**: Users must be able to register accounts with email and password authentication
- **FR-2**: Users must be able to link their Riot account (PUUID) to their platform account
- **FR-3**: System must support role-based access control (Admin, Player)
- **FR-4**: Users must be able to authenticate and maintain sessions via JWT tokens

#### League Management (Admin)

- **FR-5**: Admins must be able to create new leagues
- **FR-6**: Admins must be able to create seasons within leagues with defined start/end dates
- **FR-7**: Admins must be able to schedule series and matches for teams
- **FR-8**: Admins must be able to approve/reject team registrations for leagues
- **FR-9**: Admins must be able to import match data using Riot match IDs
- **FR-10**: Admins must be able to verify that match participants align with registered teams

#### Team Management

- **FR-11**: Players must be able to create teams (becoming team captain)
- **FR-12**: Team captains must be able to invite other players to join their team
- **FR-13**: Players must be able to accept/reject team invitations
- **FR-14**: Teams must be able to register for league seasons
- **FR-15**: Team captains must be able to manage team rosters

#### Match Management

- **FR-16**: System must support match scheduling with series structure (Bo1, Bo3, Bo5)
- **FR-17**: System must track match status (scheduled, ongoing, completed, cancelled)
- **FR-18**: System must store comprehensive match data from API responses
- **FR-19**: System must map API team colors (Red/Blue) to actual registered teams
- **FR-20**: System must handle team side swapping after round 12

#### Statistics Tracking

- **FR-21**: System must track individual player statistics per match
- **FR-22**: System must track round-by-round player performance
- **FR-23**: System must calculate derived metrics (ADR, ACS, K/D, HS%, KAST)
- **FR-24**: System must track kill events with weapon, location, and timestamp data
- **FR-25**: System must track plant/defuse events with location data
- **FR-26**: System must track economy statistics (spent, loadout values)
- **FR-27**: System must track ability usage per player

### Non-Functional Requirements

#### Performance

- **NFR-1**: API responses must complete within 2 seconds for 95% of requests
- **NFR-2**: Database queries must be optimized with appropriate indexing
- **NFR-3**: Frontend must render initial page load within 3 seconds on standard connections
- **NFR-4**: System must handle concurrent users (target: 100+ simultaneous users)

#### Security

- **NFR-5**: Passwords must be hashed using bcrypt with minimum 10 salt rounds
- **NFR-6**: JWT tokens must expire after 7 days
- **NFR-7**: API endpoints must validate and sanitize all user inputs
- **NFR-8**: Admin actions must require proper authorization checks
- **NFR-9**: Sensitive data must not be exposed in API responses

#### Scalability

- **NFR-10**: Database schema must support multiple concurrent leagues
- **NFR-11**: System must handle large match data imports (1MB+ JSON responses)
- **NFR-12**: Architecture must support horizontal scaling of backend services

#### Reliability

- **NFR-13**: System must handle API integration failures gracefully
- **NFR-14**: Database transactions must maintain ACID properties
- **NFR-15**: Error states must provide meaningful feedback to users

#### Maintainability

- **NFR-16**: Code must follow TypeScript strict mode standards

- **NFR-17**: API must follow RESTful conventions
- **NFR-18**: Code must maintain modular architecture (controller/service/repository pattern)
- **NFR-19**: Database migrations must be versioned and reversible

#### Usability

- **NFR-20**: User interface must be intuitive for non-technical users
- **NFR-21**: Error messages must be clear and actionable
- **NFR-22**: Admin workflows must minimize manual data entry

### Edge Case Considerations

#### User & Authentication

- **EC-1**: User attempts to link already-linked Riot account
- **EC-2**: User's Riot account name/tag changes after linking
- **EC-3**: Multiple users attempt to link same PUUID
- **EC-4**: User tries to access admin functions without proper role
- **EC-5**: JWT token expires during active session

#### Team Management

- **EC-6**: Player receives multiple team invitations simultaneously
- **EC-7**: Team captain leaves team before appointing new captain
- **EC-8**: Player is on multiple team rosters in same season
- **EC-9**: Team attempts to register for league after registration deadline
- **EC-10**: Team disbands after being scheduled for matches

#### Match Import & Verification

- **EC-11**: Match ID already exists in database (duplicate import)
- **EC-12**: Match participants don't match either team's roster
- **EC-13**: Match has substitute player not on official roster
- **EC-14**: API response contains incomplete or malformed data
- **EC-15**: Match was forfeited before completion
- **EC-16**: Custom game had uneven teams (4v5, etc.)
- **EC-17**: Player PUUID in API doesn't match any registered player

#### Scheduling & Series

- **EC-18**: Series scheduled with teams from different leagues
- **EC-19**: Team scheduled for overlapping matches
- **EC-20**: Match imported for wrong series
- **EC-21**: Series winner determination with tied scores

#### Statistics

- **EC-22**: Player stats contain zero division scenarios (deaths = 0 for K/D)

- **EC-23**: Negative damage values from API
- **EC-24**: Round count doesn't match expected total (e.g., overtime)
- **EC-25**: Agent ID from API doesn't exist in database

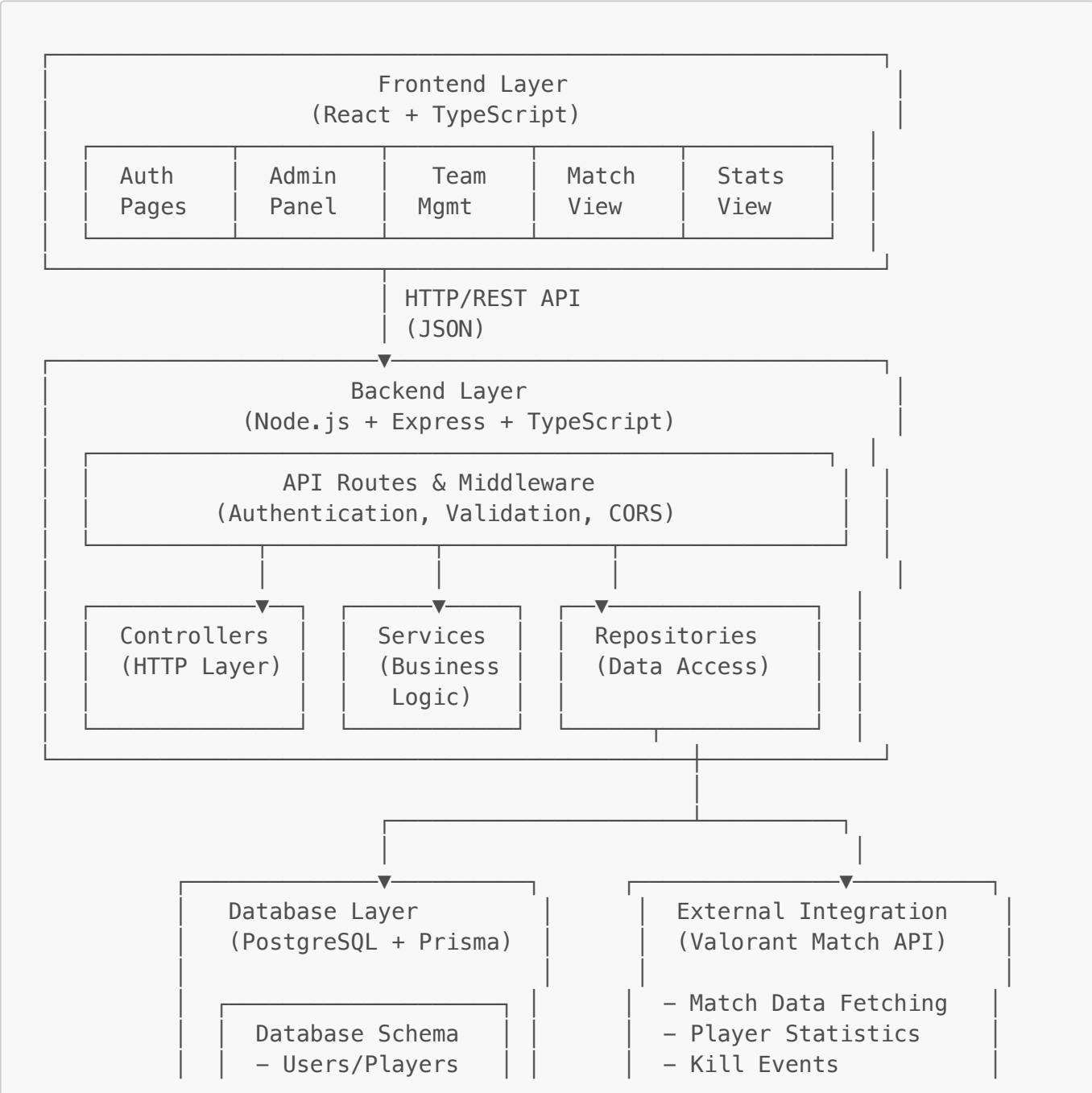
**Data Integrity**

- **EC-26**: Concurrent updates to same entity (team roster, match status)
- **EC-27**: Cascade deletions affecting related entities
- **EC-28**: Season end date before start date
- **EC-29**: Match completion timestamp before start timestamp
- **EC-30**: Network interruption during large data import

---

# System Architecture

## Core Components



```
│  │  - Teams/Rosters   │ │      │  - Round Data              │
│  │  - Leagues/Seasons │ │      │                            │
│  │  - Matches/Stats   │ │      │                            │
│  └────────────────────┘ │      └────────────────────────────┘
└─────────────────────────┘
```

## Component Analysis

### Frontend Modules

#### Public Pages

- **Landing Page**: Introduction to the platform, feature highlights
- **Login/Register**: Authentication forms with validation
- **League Browser**: Public view of active leagues and standings

#### Player Dashboard

- **Profile Management**: View/edit user profile, link Riot account
- **My Teams**: View teams user is member of
- **Team Creation**: Form to create new team
- **Team Invitations**: View and respond to team invitations
- **Team View**: Roster management, league registrations, match schedule
- **Match History**: Personal match statistics and performance
- **Statistics Dashboard**: Performance analytics and trends

#### Admin Panel

- **League Management**: Create/edit leagues and seasons
- **Team Approvals**: Review and approve team registrations
- **Match Scheduling**: Create series and schedule matches
- **Match Import**: Import match data via Riot match IDs
- **Match Verification**: Verify participants and approve match data
- **User Management**: View users, assign roles
- **System Analytics**: Platform usage statistics

#### Shared Components

- **Navigation**: Role-based navigation bar
- **Match Card**: Display match summary and stats
- **Player Card**: Display player statistics
- **Team Card**: Display team information and roster
- **Statistics Tables**: Sortable tables for various statistics
- **Forms**: Reusable form components with validation

### Backend Modules

#### Authentication Module

- **Routes**: `/api/auth/signup`, `/api/auth/login`, `/api/auth/logout`
- **Responsibilities**: User registration, login, session management
- **Security**: Password hashing, JWT generation, token validation

#### User Module

- **Routes**: `/api/users`, `/api/users/:id`, `/api/users/:id/link-riot`
- **Responsibilities**: User profile management, Riot account linking, role management

#### League Module

- **Routes**: `/api/leagues`, `/api/leagues/:id`, `/api/leagues/:id/seasons`
- **Responsibilities**: League CRUD operations, season management, registration handling

#### Team Module

- **Routes**: `/api/teams`, `/api/teams/:id`, `/api/teams/:id/roster`, `/api/teams/:id/invitations`
- **Responsibilities**: Team creation, roster management, invitation system, league registration

#### Match Module

- **Routes**: `/api/matches`, `/api/matches/:id`, `/api/matches/:id/stats`, `/api/matches/import`
- **Responsibilities**: Match scheduling, data import, verification, statistics retrieval

#### Statistics Module

- **Routes**: `/api/stats/players/:id`, `/api/stats/teams/:id`, `/api/stats/leaderboards`
- **Responsibilities**: Aggregate statistics, leaderboard generation, performance analytics

#### Reference Data Modules

- **Agents Module**: Manage Valorant agent reference data
- **Maps Module**: Manage Valorant map reference data

### Database Layer (Prisma ORM)

#### Repository Pattern

- **Purpose**: Abstraction layer between business logic and database
- **Benefits**: Testability, maintainability, consistent data access patterns
- **Structure**: One repository per entity type

#### Key Entities

- **User**: Authentication and profile data
- **Player**: Riot account information (PUUID, name, tag)
- **Team**: Team metadata and captain relationship
- **TeamRoster**: Player-team associations per season
- **League**: Top-level competition container
- **Season**: Time-bound competition period within a league
- **Series**: Best-of-N match collection between two teams
- **Match**: Individual game with statistics
- **Round**: Round-level data within a match
- **Kill/Plant/Defuse**: Event-level data
- **Statistics**: Aggregated player and team performance data

---

## High-Level Data Flow

### 1. User Registration & Riot Account Linking

```
User → Frontend → POST /api/auth/signup → Backend
                                            ↓
                                      Hash Password
                                            ↓
                                     Create User Record
                                            ↓
                                     Generate JWT Token
                                            ↓
Frontend ← JWT Token ← Response ← Backend

User → Frontend → POST /api/users/:id/link-riot → Backend
                                                    ↓
                                                Validate PUUID
                                                    ↓
                                            Check if PUUID exists
                                                    ↓
                                            Create/Update Player
                                                    ↓
                                            Link User ↔ Player
                                                    ↓
Frontend ← Updated Profile ← Response ← Backend
```

### 2. Team Creation & League Registration

```
Player → Frontend → POST /api/teams → Backend
                                        ↓
                                  Validate User
                                        ↓
                              Create Team (User = Captain)
                                        ↓
Frontend ← Team Data ← Response ← Backend
```

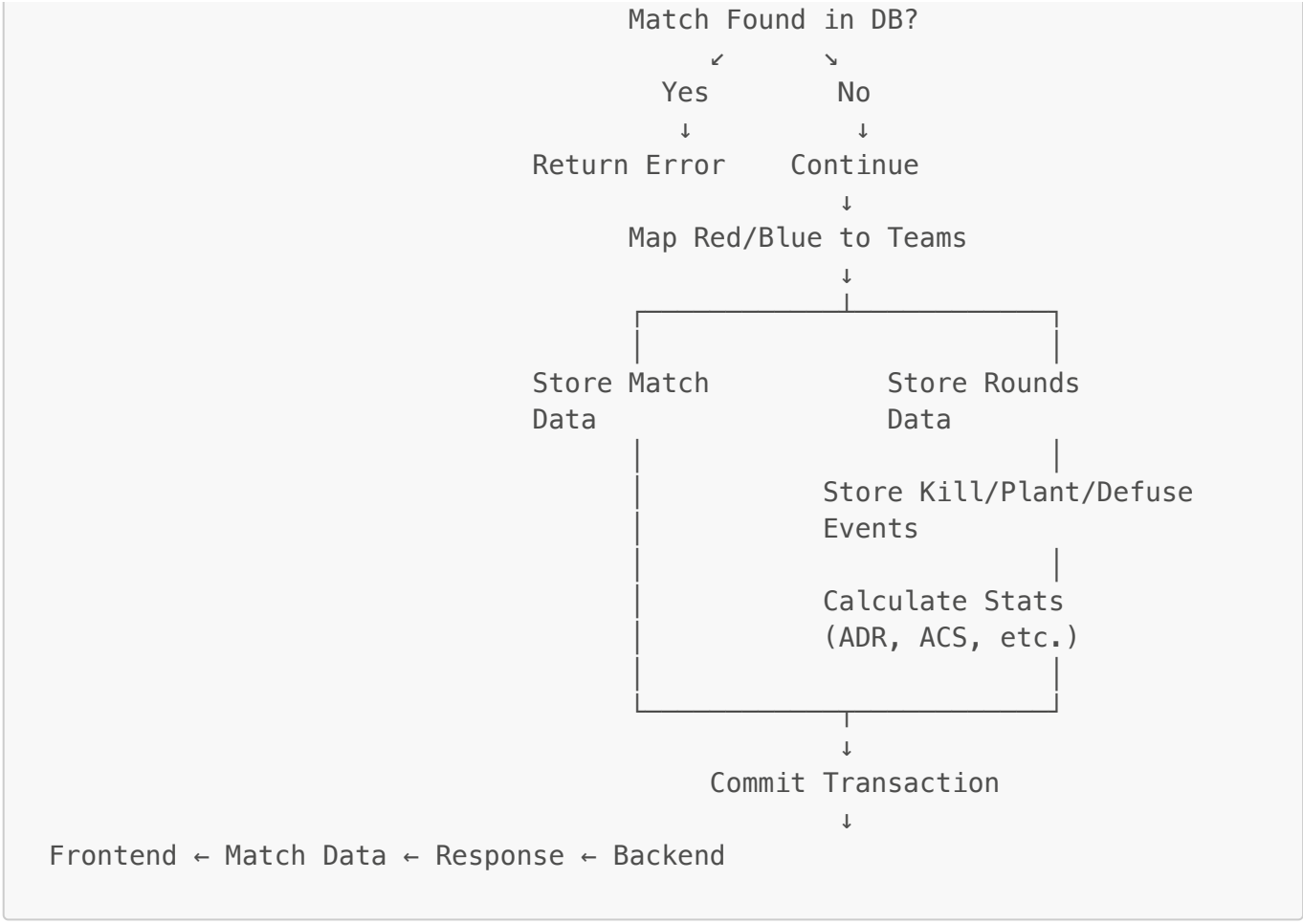
```
Captain → Frontend → POST /api/teams/:id/invitations → Backend
                                                          ↓
                                                   Validate Team
                                                          ↓
                                                 Validate Invitee
                                                          ↓
                                              Create Invitation
                                                          ↓
Frontend ← Invitation ← Response ← Backend

Captain → Frontend → POST /api/teams/:id/register → Backend
                                                        ↓
                                                  Validate Team
                                                        ↓
                                                Validate League
                                                        ↓
                                          Create Registration (Pending)
                                                        ↓
Frontend ← Registration ← Response ← Backend
```

#### 3. Admin Match Scheduling

```
Admin → Frontend → POST /api/series → Backend
                                        ↓
                                  Validate Teams
                                        ↓
                                 Validate Season
                                        ↓
                            Create Series Record
                                        ↓
                           Create Match Records
                                        ↓
Frontend ← Series Data ← Response ← Backend
```

#### 4. Match Data Import & Verification

```
Admin → Frontend → POST /api/matches/import → Backend
                    (Riot Match ID)             ↓
                                          Fetch from API
                                                ↓
                                                |
                                    ┌───────────┴──────────┐
                                    |                      |
                              Parse Match            Verify Players
                              Metadata               Match Roster
                                    |                      |
                                    └──────────┬───────────┘
                                               |
                                               ↓
```




#### 5. Statistics Retrieval

```
User → Frontend → GET /api/stats/players/:id → Backend
                                                  ↓
                                          Query Match Stats
                                                  ↓
                                        Aggregate Performance
                                                  ↓
                                          Calculate Averages
                                                  ↓
                                          Calculate Trends
                                                  ↓
Frontend ← Statistics ← Response ← Backend
```

---

## Backend API Endpoints

### Authentication Endpoints

| Method | Endpoint | Description | Auth Required |
|---|---|---|---|
| POST | `/api/auth/signup` | Register new user | No |
| POST | `/api/auth/login` | Authenticate user | No |

| Method | Endpoint | Description | Auth Required |
|---|---|---|---|
| POST | `/api/auth/logout` | End user session | Yes |

### User Endpoints

| Method | Endpoint | Description | Auth Required | Admin Only |
|---|---|---|---|---|
| GET | `/api/users` | Get all users | Yes | Yes |
| GET | `/api/users/:id` | Get user by ID | Yes | No |
| PATCH | `/api/users/:id` | Update user profile | Yes | No |
| POST | `/api/users/:id/link-riot` | Link Riot account | Yes | No |
| PATCH | `/api/users/:id/role` | Update user role | Yes | Yes |

### League Endpoints

| Method | Endpoint | Description | Auth Required | Admin Only |
|---|---|---|---|---|
| GET | `/api/leagues` | Get all leagues | No | No |
| GET | `/api/leagues/:id` | Get league by ID | No | No |
| POST | `/api/leagues` | Create new league | Yes | Yes |
| PATCH | `/api/leagues/:id` | Update league | Yes | Yes |
| DELETE | `/api/leagues/:id` | Delete league | Yes | Yes |

### Season Endpoints

| Method | Endpoint | Description | Auth Required | Admin Only |
|---|---|---|---|---|
| GET | `/api/seasons` | Get all seasons | No | No |
| GET | `/api/seasons/:id` | Get season by ID | No | No |
| GET | `/api/leagues/:id/seasons` | Get seasons for league | No | No |
| POST | `/api/seasons` | Create new season | Yes | Yes |
| PATCH | `/api/seasons/:id` | Update season | Yes | Yes |
| DELETE | `/api/seasons/:id` | Delete season | Yes | Yes |

### Team Endpoints

| Method | Endpoint | Description | Auth Required | Admin Only |
|---|---|---|---|---|
| GET | `/api/teams` | Get all teams | No | No |

| Method | Endpoint | Description | Auth Required | Admin Only |
|---|---|---|---|---|
| GET | `/api/teams/:id` | Get team by ID | No | No |
| POST | `/api/teams` | Create new team | Yes | No |
| PATCH | `/api/teams/:id` | Update team (captain only) | Yes | No |
| DELETE | `/api/teams/:id` | Delete team | Yes | No |
| GET | `/api/teams/:id/roster` | Get team roster | No | No |
| POST | `/api/teams/:id/roster` | Add player to roster | Yes | No |
| DELETE | `/api/teams/:id/roster/:playerId` | Remove player from roster | Yes | No |

**Team Invitation Endpoints**

| Method | Endpoint | Description | Auth Required |
|---|---|---|---|
| GET | `/api/invitations` | Get user's invitations | Yes |
| POST | `/api/teams/:id/invitations` | Create invitation (captain only) | Yes |
| PATCH | `/api/invitations/:id/accept` | Accept invitation | Yes |
| PATCH | `/api/invitations/:id/reject` | Reject invitation | Yes |

**League Registration Endpoints**

| Method | Endpoint | Description | Auth Required | Admin Only |
|---|---|---|---|---|
| POST | `/api/leagues/:id/register` | Register team for league | Yes | No |
| GET | `/api/leagues/:id/registrations` | Get league registrations | Yes | Yes |
| PATCH | `/api/registrations/:id/approve` | Approve registration | Yes | Yes |
| PATCH | `/api/registrations/:id/reject` | Reject registration | Yes | Yes |

**Series Endpoints**

| Method | Endpoint | Description | Auth Required | Admin Only |
|---|---|---|---|---|
| GET | `/api/series` | Get all series | No | No |
| GET | `/api/series/:id` | Get series by ID | No | No |

| Method | Endpoint | Description | Auth Required | Admin Only |
|---|---|---|---|---|
| POST | `/api/series` | Create new series | Yes | Yes |
| PATCH | `/api/series/:id` | Update series | Yes | Yes |
| DELETE | `/api/series/:id` | Delete series | Yes | Yes |

#### Match Endpoints

| Method | Endpoint | Description | Auth Required | Admin Only |
|---|---|---|---|---|
| GET | `/api/matches` | Get all matches | No | No |
| GET | `/api/matches/:id` | Get match by ID | No | No |
| POST | `/api/matches` | Create new match | Yes | Yes |
| POST | `/api/matches/import` | Import match from Riot API | Yes | Yes |
| PATCH | `/api/matches/:id/verify` | Verify match data | Yes | Yes |
| GET | `/api/matches/:id/stats` | Get match statistics | No | No |

#### Statistics Endpoints

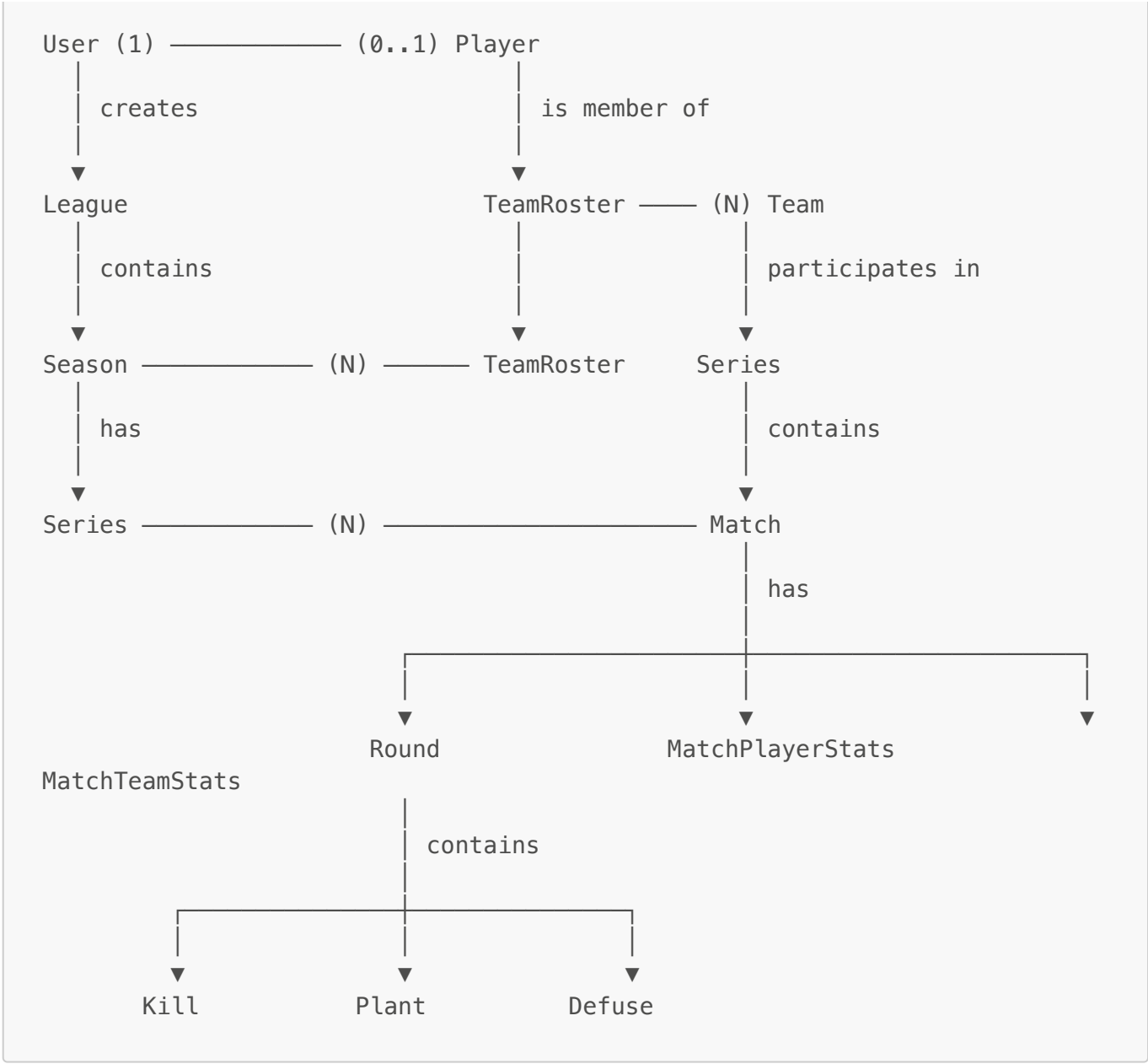
| Method | Endpoint | Description | Auth Required |
|---|---|---|---|
| GET | `/api/stats/players/:id` | Get player statistics | No |
| GET | `/api/stats/players/:id/matches` | Get player match history | No |
| GET | `/api/stats/teams/:id` | Get team statistics | No |
| GET | `/api/stats/leagues/:id/leaderboard` | Get league leaderboard | No |

#### Reference Data Endpoints

| Method | Endpoint | Description | Auth Required | Admin Only |
|---|---|---|---|---|
| GET | `/api/agents` | Get all agents | No | No |
| POST | `/api/agents` | Create agent | Yes | Yes |
| GET | `/api/maps` | Get all maps | No | No |
| POST | `/api/maps` | Create map | Yes | Yes |

## Database Schema Overview

#### Core Entity Relationships




### Key Entities

#### User Management

- **User**: Authentication credentials, role, profile data
- **Player**: Riot account information (PUUID, name#tag)

#### League Structure

- **League**: Top-level competition entity
- **Season**: Time-bound period within a league
- **LeagueRegistration**: Team registration for league/season

#### Team Management

- **Team**: Team metadata, captain relationship
- **TeamRoster**: Player memberships per season
- **TeamInvitation**: Pending team invitations

#### Match Structure

- **Series**: Collection of matches (Bo1/3/5) between two teams
- **Match**: Single game on a specific map
- **MatchParticipation**: Player participation tracking

#### Event Data

- **Round**: Round-level data within a match
- **Kill**: Kill event with weapon, location, timestamp
- **Plant**: Spike plant event with location
- **Defuse**: Spike defuse event with location

#### Statistics

- **MatchPlayerStats**: Aggregated player stats per match
- **MatchTeamStats**: Aggregated team stats per match
- **RoundPlayerStats**: Per-round player performance
- **RoundTeamStats**: Per-round team performance

#### Reference Data

- **Agent**: Valorant character reference
- **Map**: Valorant map reference

### Key Schema Constraints

#### Uniqueness Constraints

- User: email, username
- Player: puuid, (name + tag)
- Team: name
- Agent: name, id
- Map: name, id
- Match: riotMatchId
- LeagueRegistration: (leagueId + teamId + seasonId)

#### Required Relationships

- Match → Series → Season → League
- MatchPlayerStats → Player, Match
- TeamRoster → Team, Player, Season
- Kill → Killer (Player), Victim (Player), Match
- Team → Captain (Player)

#### Cascade Behaviors

- Delete Match → Delete Rounds, Stats, Events

- Delete Team → Update/Nullify related records
- Delete Season → Preserve historical data

---

# Development Roadmap

## Phase 1: Foundation & Authentication

**Objectives**: Establish core infrastructure and user management

### Database & Backend Setup

- Initialize Prisma schema with Phase 1 entities
- Create database migrations for User and Player tables
- Set up user roles enum (admin, player)
- Implement database indexing strategy

### Authentication System

- Implement user registration endpoint
- Implement login endpoint with JWT generation
- Create authentication middleware for protected routes
- Implement password hashing with bcrypt
- Create role-based authorization middleware

### User Profile Management

- Create user profile endpoints (GET, PATCH)
- Implement Riot account linking endpoint
- Add validation for PUUID uniqueness
- Create player profile retrieval endpoints

### Frontend - Authentication

- Build login page component
- Build registration page component
- Implement JWT token storage and management
- Create authentication context/provider
- Build protected route wrapper
- Create user profile page

**Deliverables**:

- Functional user registration and login
- Riot account linking capability
- Role-based access control foundation
- User profile management

---

## Phase 2: League & Team Management

**Objectives**: Enable league creation and team formation

**Database Extensions**

- Add League, Season, Team entities
- Add TeamInvitation, LeagueRegistration entities
- Create appropriate indexes and relationships
- Implement cascade deletion rules

**League Administration (Backend)**

- Create league CRUD endpoints
- Create season CRUD endpoints
- Implement league registration approval endpoints
- Add validation for date ranges and conflicts

**Team Management (Backend)**

- Create team creation endpoint (sets creator as captain)
- Implement team invitation system endpoints
- Create team roster management endpoints
- Implement league registration endpoints
- Add validation for team constraints

**Frontend - Admin Panel**

- Build league creation form
- Build season management interface
- Create league registration approval interface
- Build admin dashboard with league overview

**Frontend - Team Management**

- Build team creation form
- Create team dashboard/profile page
- Implement team invitation interface
- Build roster management interface
- Create league registration form
- Build "My Teams" overview page

**Deliverables**:

- Admins can create and manage leagues/seasons
- Players can create teams and become captains
- Team invitation and acceptance workflow
- Team registration for leagues
- Admin approval workflow for registrations

---

## Phase 3: Match Scheduling & Series Management

**Objectives**: Enable match scheduling and series organization

### Database Extensions

- Add Series and Match entities with Red/Blue team mapping
- Add match status tracking fields
- Create submission and verification tracking fields
- Implement match-team relationships

### Series Management (Backend)

- Create series CRUD endpoints
- Implement match creation within series
- Add validation for team eligibility
- Create match scheduling endpoints

### Match Management (Backend)

- Implement match status update endpoints
- Create match retrieval endpoints with filters
- Add series-match relationship queries
- Implement match verification endpoints

### Frontend - Admin Scheduling

- Build series creation interface
- Create match scheduling calendar/timeline view
- Implement match status management interface
- Build team selection interface for series

### Frontend - Match Views

- Create match schedule view (public)
- Build match details page (public)
- Implement team schedule view
- Create match management interface (admin)

**Deliverables**:

- Admins can create series and schedule matches
- Teams can view their match schedule
- Public match schedule accessible
- Match status tracking (scheduled → ongoing → completed)

---

## Phase 4: API Integration & Match Import

**Objectives**: Integrate with Valorant match API and import match data

**External API Integration**

- Create API client service for Valorant match data
- Implement API error handling and retry logic
- Add rate limiting considerations
- Create data transformation layer (API → Database schema)

**Match Import (Backend)**

- Create match import endpoint
- Implement match data parsing from API response
- Build Red/Blue team to registered team mapping logic
- Create player verification against rosters
- Implement duplicate match detection
- Add comprehensive error handling for import failures

**Match Verification (Backend)**

- Create participant verification logic
- Implement roster mismatch detection and reporting
- Add admin override capabilities
- Create verification approval endpoints

**Statistics Calculation**

- Implement ADR, ACS, K/D calculation logic
- Create headshot percentage calculation
- Implement KAST calculation (requires round analysis)
- Add first blood/death tracking

**Frontend - Match Import**

- Build match import interface (admin)
- Create verification review page
- Implement mismatch reporting UI
- Build match import status tracker
- Create error handling and user feedback

**Deliverables**:

- Admins can import matches via Riot match ID
- System validates participants against rosters
- Automatic calculation of advanced statistics
- Admin can review and approve/reject imports
- Error handling for edge cases (substitutes, mismatches)

---

## Phase 5: Data Storage & Event Tracking

**Objectives**: Store comprehensive match data and events

### Database Extensions

- Add Round, Kill, Plant, Defuse entities
- Add MatchPlayerStats, RoundPlayerStats entities
- Add MatchTeamStats, RoundTeamStats entities
- Create MatchParticipation tracking
- Implement event-level indexing

### Event Storage (Backend)

- Create endpoints for storing round data
- Implement kill event storage with locations
- Add plant/defuse event storage
- Create round-level statistics storage
- Implement match-level statistics aggregation

### Statistics Aggregation

- Create batch statistics calculation
- Implement player statistics aggregation
- Add team statistics aggregation
- Create leaderboard calculation logic

**Deliverables**:

- Complete match data storage (rounds, events, stats)
- Round-by-round data tracking
- Kill events with weapon and location data
- Plant/defuse tracking
- Aggregated match statistics

---

## Phase 6: Statistics & Analytics

**Objectives**: Display comprehensive statistics and analytics

### Statistics Endpoints (Backend)

- Create player statistics endpoints (overall, per season, per match)
- Implement team statistics endpoints
- Build leaderboard endpoints (kills, ADR, ACS, etc.)
- Add head-to-head comparison endpoints
- Create trend analysis endpoints

### Analytics Calculations

- Implement performance trend calculations
- Add win/loss ratio calculations
- Create agent usage statistics
- Calculate map performance statistics
- Implement time-based aggregations

**Frontend - Player Statistics**

- Build player profile statistics page
- Create player match history view
- Implement performance trend charts
- Build agent usage breakdown
- Create comparison interface

**Frontend - Team Statistics**

- Build team statistics dashboard
- Create team match history view
- Implement roster performance breakdown
- Build win/loss visualization

**Frontend - Leaderboards**

- Create league leaderboard view
- Implement sortable statistics tables
- Build filter interface (season, role, map, etc.)
- Create statistical rankings

**Deliverables**:

- Comprehensive player statistics pages
- Team performance analytics
- League leaderboards
- Performance trends and visualizations
- Statistical comparisons

---

## Phase 7: Reference Data & System Setup

**Objectives**: Populate and manage reference data

**Reference Data Management**

- Create agent seeding script from official data
- Create map seeding script from official data
- Implement agent/map admin interfaces
- Add reference data update endpoints

**System Initialization**

- Create database seeding scripts
- Implement initial admin user creation
- Add system configuration management
- Create data backup procedures

**Frontend - Reference Data**

- Build agent reference page
- Create map reference page
- Implement admin reference data management

**Deliverables**:

- Complete agent and map reference data
- Admin interfaces for reference data
- System initialization procedures

---

## Phase 8: Polish & Optimization

**Objectives**: Refine user experience and optimize performance

**Performance Optimization**

- Implement database query optimization
- Add API response caching
- Optimize statistics calculations
- Implement pagination for large datasets
- Add database connection pooling

**User Experience**

- Refine UI/UX based on testing
- Add loading states and skeletons
- Implement error boundaries
- Enhance form validation feedback
- Add confirmation dialogs for destructive actions

**Security Hardening**

- Implement rate limiting
- Add input sanitization
- Create audit logging for admin actions
- Implement CSRF protection
- Add security headers

**Testing**

- Create integration tests for critical paths

- Add end-to-end tests for workflows
- Implement API endpoint testing
- Create database migration testing

**Deliverables**:

- Optimized application performance
- Enhanced user experience
- Hardened security
- Comprehensive testing coverage

---

## Phase 9: Documentation & Deployment

**Objectives**: Prepare for production deployment

**Documentation**

- Create API documentation (OpenAPI/Swagger)
- Write deployment guide
- Create admin user guide
- Write player user guide
- Document database schema

**Deployment Preparation**

- Set up production database
- Configure environment variables
- Implement database migration strategy
- Create backup and recovery procedures
- Set up monitoring and logging

**Production Deployment**

- Deploy backend services
- Deploy frontend application
- Configure domain and SSL
- Implement health checks
- Set up error monitoring

**Deliverables**:

- Complete system documentation
- Production-ready deployment
- Monitoring and logging in place
- Backup and recovery procedures

---

## Phase 10: Future Enhancements

**Objectives**: Advanced features and improvements

**Potential Enhancements**

- Player-submitted match imports (with admin approval)
- Match code system for automatic detection
- Email notifications for invitations and matches
- Advanced analytics and AI-powered insights
- Mobile responsive design improvements
- Social features (comments, reactions)
- Tournament bracket visualization
- Player achievements and badges
- Export statistics to CSV/PDF
- Integration with Discord for notifications

---

# Appendix

## Technology Versions

- Node.js: 18+
- TypeScript: 5.x
- React: 18+
- Express: 5.x
- Prisma: 7.x
- PostgreSQL: 14+

## Development Environment Setup

1. Install Node.js and npm
2. Install PostgreSQL
3. Clone repository
4. Install dependencies: `npm install`
5. Set up environment variables (.env)
6. Run database migrations: `npx prisma migrate dev`
7. Seed reference data: `npm run seed`
8. Start development server: `npm run dev`

## Environment Variables

```
# Database
DATABASE_URL="postgresql://user:password@localhost:5432/valorant_league"

# Authentication
JWT_SECRET="your-secret-key-change-in-production"
JWT_EXPIRES_IN="7d"

# Frontend
FRONTEND_URL="http://localhost:5173"
```

```
# API
PORT=5001

# External API
VALORANT_API_URL="https://api.example.com"
VALORANT_API_KEY="your-api-key"
```

---

**Document Version**: 1.0 **Last Updated**: December 2024 **Author**: Development Team