# Comprehensive SDLC Plan for Rowdy Cup Tournament Management Application

## Summary

This document outlines a comprehensive Software Development Life Cycle (SDLC) plan for the Rowdy Cup tournament management application. The application is a mobile-first Progressive Web App (PWA) built with SvelteKit and Supabase, designed to provide real-time tournament status, scorekeeping, and a betting platform for users. This SDLC plan covers all phases from project initiation through maintenance and support, providing detailed guidance for the development team.

## 1. Project Initiation & Planning

### 1.1 Project Overview and Objectives

**Project Name:** Rowdy Cup Tournament Management Application

**Project Description:** A mobile-first Progressive Web App (PWA) that provides comprehensive tournament management capabilities for the Rowdy Cup golf tournament series, including real-time tournament status, scorekeeping, and a betting platform.

**Primary Objectives:**

* Create a user-friendly mobile application for tournament management
* Enable real-time tournament status updates and scorekeeping
* Implement a betting platform for match and player performance
* Support offline functionality for score entry and data access
* Provide comprehensive tournament statistics and historical data

### 1.2 Scope Definition

**In Scope:**

* User authentication with username and 4-digit PIN
* Tournament, team, and player management
* Multiple match format support (2v2 Team Scramble, 2v2 Team Shamble, 4v4 Team Scramble, 2v2 Team Best Ball, and 1v1 Individual Match Play)
* Real-time score entry and updates
* Offline support with data synchronization
* Sportsbook module for betting
* Statistics, standings, and historical data
* Mobile-responsive design with top-nav hamburger menu
* Row-Level Security (RLS) implementation

**Out of Scope:**

* Native mobile applications (iOS/Android)
* Payment processing for betting (tracking only)
* Third-party integrations beyond Supabase
* Automated video/media content generation

### 1.5 Resource Planning

**Infrastructure Resources:**

* Supabase (Backend as a Service)
* Vercel (Hosting)
* GitHub (Version Control and CI/CD)
* Development, Testing, and Production environments

**Tools:**

* SvelteKit for frontend and API routes
* Supabase for database, authentication, and real-time updates
* Vitest for unit testing
* GitHub Actions for CI/CD
* ESLint and Prettier for code quality

## 2. Requirements Analysis

### 2.1 Functional Requirements

#### 2.1.1 User Authentication and Authorization

* Admin-only user creation with username and 4-digit PIN login
* Role-based access control (admin, player, public)
* Protected routes with redirect-to pattern
* Session management and persistence

#### 2.1.2 Tournament Management

* Create, edit, and delete tournaments
* Define tournament location, year, and name
* Assign teams to tournaments
* Configure tournament rules and formats

#### 2.1.3 Team and Player Management

* Create and manage teams (two teams per tournament)
* Add, edit, and remove players
* Assign players to teams
* Track player statistics and history

#### 2.1.4 Round and Match Management

* Support for multiple match formats:
  + 2v2 Team Scramble
  + 2v2 Team Shamble
  + 4v4 Team Scramble
  + 2v2 Team Best Ball
  + 1v1 Individual Match Play
* Schedule rounds and matches
* Define scoring rules for each match type
* Track match status (pending, in progress, completed)

#### 2.1.5 Score Entry and Tracking

* Mobile-friendly score entry interface
* Real-time score updates
* Net scoring and handicap calculations
* Offline score entry with synchronization
* Score validation and verification

#### 2.1.6 Statistics and Reporting

* Compute win/loss/tie records
* Track head-to-head statistics
* Generate leaderboards
* Display historical performance data
* Export tournament results

#### 2.1.7 Sportsbook Module

* Place bets on matches and player performance
* Track bet status and outcomes
* Automatic bet resolution upon match completion
* Real-money tracking (without payment processing)
* Betting history and statistics

#### 2.1.8 Offline Support

* Cache tournament data for offline access
* Queue score entries for later synchronization
* Offline authentication
* Synchronization status indicators
* Conflict resolution for offline changes

### 2.2 Non-Functional Requirements

#### 2.2.1 Performance

* Page load time under 2 seconds on 4G connections
* Real-time updates delivered within 3 seconds
* Support for up to 100 concurrent users
* Efficient data loading and pagination

#### 2.2.2 Security

* Row-Level Security (RLS) implementation
* Secure authentication with PIN hashing
* Protection against common web vulnerabilities (XSS, CSRF)
* Data encryption for sensitive information
* Regular security audits

#### 2.2.3 Usability

* Mobile-first responsive design
* Intuitive navigation with top-nav hamburger menu
* Consistent UI patterns across the application
* Clear error messages and feedback
* Accessibility compliance (WCAG 2.1 AA)

#### 2.2.4 Reliability

* 99.9% uptime during tournament hours
* Graceful degradation during connectivity issues
* Data backup and recovery procedures
* Error logging and monitoring

#### 2.2.5 Scalability

* Support for multiple concurrent tournaments
* Efficient database queries and indexing
* Optimized real-time subscription management
* Caching strategies for frequently accessed data

### 2.3 User Stories and Use Cases

#### Admin User Stories

1. As an admin, I want to create new tournaments so that I can organize events
2. As an admin, I want to create user accounts so that players can access the system
3. As an admin, I want to manage teams and players so that I can organize tournament participation
4. As an admin, I want to schedule rounds and matches so that players know when and who they're playing
5. As an admin, I want to view and edit all scores so that I can correct any errors

#### Player User Stories

1. As a player, I want to log in with a simple username and PIN so that I can quickly access the app
2. As a player, I want to view my upcoming matches so that I know my schedule
3. As a player, I want to enter scores for my matches so that results are recorded accurately
4. As a player, I want to view tournament standings so that I know my position
5. As a player, I want to place bets on matches so that I can participate in the sportsbook

#### Public User Stories

1. As a spectator, I want to view tournament information without logging in so that I can follow the event
2. As a spectator, I want to see real-time score updates so that I can stay informed about match progress
3. As a spectator, I want to view historical tournament data so that I can understand past performance

### 2.4 Data Requirements

#### Core Data Entities

1. **Tournaments**
   * ID, name, year, location, created\_at
2. **Teams**
   * ID, tournament\_id, name
3. **Players**
   * ID, team\_id, full\_name, email, created\_at
4. **Rounds**
   * ID, tournament\_id, format, sequence
5. **Matches**
   * ID, round\_id, team\_a\_id, team\_b\_id, scheduled\_at, status
6. **Scores**
   * ID, match\_id, player\_id, points\_awarded, created\_at
7. **Head-to-Head Records**
   * Player/team statistics against specific opponents
8. **User Profiles**
   * ID, username, role (admin, player), PIN (hashed)
9. **Bets**
   * ID, user\_id, match\_id, bet\_type, amount, odds, status

### 2.5 Security Requirements

1. **Authentication**
   * Username and 4-digit PIN login
   * Secure PIN storage (hashed, not plaintext)
   * Session management and timeout
2. **Authorization**
   * Row-Level Security (RLS) policies for data access
   * Public read access to tournament and team data
   * Restricted score entry to participating players and administrators
   * Admin-only access to user management and tournament configuration
3. **Data Protection**
   * Secure API endpoints
   * Input validation and sanitization
   * Protection against injection attacks
   * Secure environment variable handling

### 2.6 Constraints and Assumptions

**Constraints:**

* Vercel free tier limitations for hosting
* Supabase service limitations
* Mobile device capabilities and network conditions
* Browser compatibility requirements

**Assumptions:**

* Users will primarily access the application via mobile devices
* Internet connectivity may be intermittent during tournaments
* Tournament administrators will have basic technical proficiency
* Maximum of 100 concurrent users during peak tournament times

## 3. System Design

### 3.1 Architecture Design

#### 3.1.1 High-Level Architecture

The application will follow a client-server architecture using SvelteKit for the frontend and server components, with Supabase providing backend services:

* **Frontend:** SvelteKit (SSR and client-side rendering)
* **Backend Services:** Supabase (PostgreSQL, Auth, Realtime)
* **Hosting:** Vercel (frontend), Supabase (backend)
* **PWA Features:** Service worker for offline support, caching

#### 3.1.2 Component Diagram

+-------------------+ +-------------------+ +-------------------+  
| | | | | |  
| SvelteKit App |<--->| SvelteKit API |<--->| Supabase |  
| (Client-side) | | (Server-side) | | Services |  
| | | | | |  
+-------------------+ +-------------------+ +-------------------+  
 ^ ^  
 | |  
 v v  
+-------------------+ +-------------------+  
| | | |  
| Service Worker | | PostgreSQL DB |  
| (Offline Support)| | |  
| | | |  
+-------------------+ +-------------------+

#### 3.1.3 Technology Stack

* **Frontend Framework:** SvelteKit
* **UI Components:** Custom Svelte components
* **State Management:** Svelte stores
* **API Communication:** SvelteKit endpoints, Supabase client
* **Database:** PostgreSQL (via Supabase)
* **Authentication:** Supabase Auth (customized for username/PIN)
* **Real-time Updates:** Supabase Realtime
* **Offline Support:** Service worker, IndexedDB
* **Testing:** Vitest, Playwright
* **CI/CD:** GitHub Actions
* **Hosting:** Vercel (frontend), Supabase (backend)

### 3.2 Database Schema Design

#### 3.2.1 Entity-Relationship Diagram

The database schema will include the following tables and relationships:

* **tournaments** (id, name, year, location, created\_at)
* **teams** (id, tournament\_id, name)
* **players** (id, team\_id, full\_name, email, created\_at)
* **rounds** (id, tournament\_id, format, sequence)
* **matches** (id, round\_id, team\_a\_id, team\_b\_id, scheduled\_at, status)
* **scores** (id, match\_id, player\_id, points\_awarded, created\_at)
* **profiles** (id, username, role, pin\_hash)
* **bets** (id, user\_id, match\_id, bet\_type, amount, odds, status)

#### 3.2.2 Row-Level Security Policies

RLS policies will be implemented to restrict data access based on user roles:

1. **Public Access Policies:**
   * Allow public read access to tournaments, teams, rounds, and matches
   * Restrict write access to authenticated users only
2. **Player Access Policies:**
   * Allow players to read all tournament data
   * Allow players to enter scores only for matches they participate in
   * Allow players to view and place bets
3. **Admin Access Policies:**
   * Full CRUD access to all tables
   * Ability to create and manage user accounts
   * Ability to override scores and resolve disputes

### 3.3 UI/UX Design

#### 3.3.1 Design Principles

* Mobile-first responsive design
* Simple and intuitive interface
* Consistent navigation and interaction patterns
* Clear visual hierarchy and feedback
* Accessibility compliance

#### 3.3.2 Navigation Structure

* Top navigation bar with hamburger menu
* Main sections:
  + Dashboard/Home
  + Tournaments
  + Matches
  + Leaderboard
  + Sportsbook
  + Profile/Settings

#### 3.3.3 Key UI Components

1. **Tournament View**
   * Tournament details and schedule
   * Team listings and standings
   * Round and match navigation
2. **Match Scoring Interface**
   * Mobile-optimized score entry
   * Real-time validation and feedback
   * Offline indicator and sync status
3. **Leaderboard**
   * Sortable and filterable standings
   * Visual indicators for position changes
   * Historical performance data
4. **Sportsbook Interface**
   * Available bets listing
   * Bet placement form
   * Bet history and status tracking
5. **Admin Dashboard**
   * User management
   * Tournament configuration
   * System status and monitoring

#### 3.3.4 Responsive Design Breakpoints

* Mobile: 320px - 480px
* Tablet: 481px - 768px
* Desktop: 769px and above

### 3.4 API Design

#### 3.4.1 SvelteKit API Routes

* /api/auth/\* - Authentication endpoints
* /api/tournaments/\* - Tournament management
* /api/teams/\* - Team management
* /api/players/\* - Player management
* /api/matches/\* - Match and scoring endpoints
* /api/bets/\* - Sportsbook and betting endpoints

#### 3.4.2 Supabase API Integration

* Direct database access via Supabase client
* Real-time subscriptions for live updates
* Row-level security enforcement
* Storage API for media content

#### 3.4.3 API Security

* Server-side validation of all inputs
* Authentication checks on protected endpoints
* Rate limiting to prevent abuse
* CORS configuration

### 3.5 Security Design

#### 3.5.1 Authentication Flow

1. User enters username and 4-digit PIN
2. Client hashes PIN before transmission
3. Server validates credentials against stored hash
4. JWT token issued for authenticated session
5. Token stored securely in browser storage
6. Token refreshed automatically when needed

#### 3.5.2 Row-Level Security Implementation

* Define RLS policies in Supabase for each table
* Enforce access control based on user role and relationship to data
* Test policies thoroughly to ensure proper isolation

#### 3.5.3 Data Protection

* Sensitive data encrypted at rest
* Secure transmission over HTTPS
* Environment variables for sensitive configuration
* Regular security audits and updates

### 3.6 Offline Support Design

#### 3.6.1 Service Worker Strategy

* Cache core application assets
* Cache API responses for offline access
* Background sync for deferred operations
* Clear update and refresh mechanisms

#### 3.6.2 Data Synchronization

* Queue modifications made offline
* Sync when connectivity is restored
* Conflict resolution for concurrent changes
* User feedback during synchronization

#### 3.6.3 Offline Authentication

* Cache authentication state
* Limited offline functionality based on cached permissions
* Re-authentication when connectivity is restored

## 4. Implementation

### 4.1 Development Environment Setup

#### 4.1.1 Local Development Environment

* Node.js and npm setup
* SvelteKit project initialization
* Supabase CLI installation and configuration
* Local database setup and seeding
* Environment variable configuration

#### 4.1.2 Version Control Setup

* GitHub repository initialization
* Branch strategy (main, development, feature branches)
* Pull request templates and guidelines
* Commit message conventions

#### 4.1.3 CI/CD Pipeline Configuration

* GitHub Actions workflow setup
* Automated testing on pull requests
* Deployment to staging and production environments
* Environment-specific configuration

### 4.2 Coding Standards and Guidelines

#### 4.2.1 JavaScript/TypeScript Standards

* TypeScript for type safety
* ESLint configuration for code quality
* Prettier for consistent formatting
* JSDoc comments for documentation

#### 4.2.2 Svelte Component Guidelines

* Component organization and structure
* State management best practices
* Reactive declarations and stores
* Accessibility requirements

#### 4.2.3 CSS/Styling Approach

* CSS methodology (BEM, SMACSS, or similar)
* Mobile-first responsive design
* Theme variables for consistent styling
* Accessibility considerations

#### 4.2.4 Testing Standards

* Unit test coverage requirements
* Integration test scenarios
* End-to-end test cases
* Performance testing thresholds

### 4.3 Version Control Strategy

#### 4.3.1 Branching Strategy

* main - Production-ready code
* development - Integration branch for features
* feature/\* - Individual feature branches
* bugfix/\* - Bug fix branches
* release/\* - Release preparation branches

#### 4.3.2 Pull Request Process

1. Create feature branch from development
2. Implement and test changes
3. Submit pull request with description
4. Code review by at least one team member
5. Pass automated tests
6. Merge to development branch

#### 4.3.3 Release Process

1. Create release branch from development
2. Perform final testing and bug fixes
3. Update version numbers and documentation
4. Merge to main branch
5. Tag release with version number
6. Deploy to production

### 4.4 Implementation Phases and Priorities

#### 4.4.1 Phase 1: Core Infrastructure (Weeks 1-4)

* Project setup and configuration
* Database schema implementation
* Authentication system
* Basic UI framework and navigation

#### 4.4.2 Phase 2: Tournament Management (Weeks 5-8)

* Tournament CRUD operations
* Team and player management
* Round and match scheduling
* Basic scoring functionality

#### 4.4.3 Phase 3: Advanced Features (Weeks 9-12)

* Match format-specific scoring
* Real-time updates
* Statistics and leaderboards
* Sportsbook module

#### 4.4.4 Phase 4: Offline Support and Refinement (Weeks 13-16)

* Service worker implementation
* Offline data synchronization
* Performance optimization
* Final testing and bug fixes

### 4.5 Third-Party Integrations

#### 4.5.1 Supabase Integration

* Database connection and configuration
* Authentication setup
* Real-time subscriptions
* Storage for media content

#### 4.5.2 PWA Features

* Service worker registration
* Manifest configuration
* Install prompts and offline indicators
* Push notifications (if required)

#### 4.5.3 Analytics Integration

* Basic usage tracking
* Error reporting
* Performance monitoring
* User behavior analysis

## 5. Testing

### 5.1 Test Strategy

#### 5.1.1 Testing Levels

* Unit Testing: Individual components and functions
* Integration Testing: Component interactions
* System Testing: End-to-end functionality
* Acceptance Testing: User requirements validation

#### 5.1.2 Testing Approaches

* Test-Driven Development (TDD) for core functionality
* Behavior-Driven Development (BDD) for user-facing features
* Exploratory testing for edge cases
* Regression testing for bug fixes

#### 5.1.3 Testing Tools

* Vitest for unit testing
* Playwright for end-to-end testing
* GitHub Actions for CI testing
* Manual testing for UX validation

### 5.2 Test Cases

#### 5.2.1 Unit Test Cases

* Authentication functions
* Score calculation algorithms
* Data transformation utilities
* Form validation logic
* State management

#### 5.2.2 Integration Test Cases

* API endpoint interactions
* Database operations
* Real-time update propagation
* Authentication flow

#### 5.2.3 System Test Cases

* Tournament creation and management
* Match scoring and updates
* Leaderboard generation
* Betting placement and resolution
* Offline functionality and synchronization

#### 5.2.4 Acceptance Test Cases

* User registration and login
* Tournament navigation and viewing
* Score entry and submission
* Bet placement and tracking
* Administrative functions

### 5.3 Test Environments

#### 5.3.1 Development Environment

* Local development setup
* Mock data for testing
* Isolated database instance
* Manual testing during development

#### 5.3.2 Testing Environment

* Dedicated testing infrastructure
* Test database with sample data
* Automated test execution
* Performance testing tools

#### 5.3.3 Staging Environment

* Production-like configuration
* Data similar to production
* Pre-release testing
* User acceptance testing

#### 5.3.4 Production Environment

* Live application monitoring
* Smoke tests after deployment
* Performance monitoring
* Error tracking

### 5.4 Performance Testing

#### 5.4.1 Load Testing

* Simulate concurrent user access
* Measure response times under load
* Identify bottlenecks
* Establish performance baselines

#### 5.4.2 Stress Testing

* Test system limits
* Evaluate degradation behavior
* Recovery testing
* Resource utilization monitoring

#### 5.4.3 Mobile Performance Testing

* Various device profiles
* Network condition simulation
* Battery usage monitoring
* Offline mode performance

### 5.5 Security Testing

#### 5.5.1 Authentication Testing

* Credential validation
* Session management
* Password policies
* Brute force protection

#### 5.5.2 Authorization Testing

* Role-based access control
* Row-level security enforcement
* Permission boundaries
* Privilege escalation attempts

#### 5.5.3 Vulnerability Testing

* Common web vulnerabilities (OWASP Top 10)
* API security
* Data validation
* Secure configuration

### 5.6 User Acceptance Testing

#### 5.6.1 UAT Planning

* Test scenarios based on user stories
* Acceptance criteria definition
* User involvement scheduling
* Feedback collection methods

#### 5.6.2 UAT Execution

* Guided testing sessions
* Unguided exploration
* Feedback documentation
* Issue prioritization

#### 5.6.3 UAT Feedback Integration

* Issue triage and prioritization
* UX improvement opportunities
* Feature enhancement requests
* Documentation updates

## 6. Deployment

### 6.1 Deployment Strategy

#### 6.1.1 Deployment Environments

* Development: For ongoing development
* Staging: For pre-release testing
* Production: Live environment for users

#### 6.1.2 Deployment Process

1. Build application for target environment
2. Run pre-deployment tests
3. Deploy to target environment
4. Run post-deployment verification
5. Monitor for issues

#### 6.1.3 Deployment Frequency

* Development: Continuous deployment
* Staging: After feature completion
* Production: Scheduled releases

### 6.2 Environment Configuration

#### 6.2.1 Environment Variables

* Supabase URL and API keys
* Authentication settings
* Feature flags
* Logging and monitoring configuration

#### 6.2.2 Build Configuration

* Environment-specific builds
* Asset optimization
* Code splitting
* Service worker configuration

#### 6.2.3 Infrastructure Setup

* Vercel project configuration
* Supabase project settings
* Domain and SSL configuration
* CDN and caching settings

### 6.3 Data Migration

#### 6.3.1 Database Migration Strategy

* Schema version control
* Data migration scripts
* Rollback procedures
* Data validation

#### 6.3.2 Initial Data Setup

* Reference data population
* Test data for validation
* Admin user creation
* System configuration

#### 6.3.3 Production Data Management

* Backup procedures
* Data archiving strategy
* Database maintenance
* Performance optimization

### 6.4 Rollback Plan

#### 6.4.1 Rollback Triggers

* Critical bugs in production
* Performance degradation
* Security vulnerabilities
* Data integrity issues

#### 6.4.2 Rollback Procedure

1. Identify rollback need
2. Communicate to stakeholders
3. Deploy previous stable version
4. Verify system functionality
5. Communicate resolution

#### 6.4.3 Rollback Testing

* Regular rollback drills
* Automated rollback scripts
* Database rollback verification
* Service continuity testing

### 6.5 Post-Deployment Verification

#### 6.5.1 Smoke Testing

* Critical path verification
* Core functionality testing
* Integration point validation
* Performance spot-checks

#### 6.5.2 Monitoring Setup

* Error tracking configuration
* Performance monitoring
* User activity tracking
* System health dashboards

#### 6.5.3 Initial User Feedback

* Feedback collection mechanisms
* Issue reporting channels
* Priority assessment
* Response procedures

## 9. Conclusion

This comprehensive SDLC plan provides a detailed roadmap for the development, deployment, and maintenance of the Rowdy Cup tournament management application. By following this plan, the development team will be able to create a robust, user-friendly application that meets all the specified requirements while maintaining high standards of quality, security, and performance.

The mobile-first PWA built with SvelteKit and Supabase will provide tournament administrators and players with a powerful tool for managing tournaments, tracking scores in real-time, and engaging with the betting platform. The implementation of offline support will ensure the application remains functional even in areas with limited connectivity, while the Row-Level Security implementation will protect sensitive data and ensure appropriate access controls.

Regular reviews of this SDLC plan throughout the project lifecycle will help ensure that the development team stays on track and that any necessary adjustments can be made in a timely manner.