JustScore Technical Documentation

Tech Stack Overview

JustScore is a performance management platform built with the following technology stack:

Category	Technologies
Frontend Framework	Next.js 14.2.17, React 18
Database	PostgreSQL via Prisma ORM
Authentication	Clerk
Styling	Tailwind CSS 4.0, shadcn/ui components
State Management	Zustand, React Query (TanStack Query)
API	Next.js API Routes, RESTful design
Development Tools	TypeScript, ESLint, Storybook
Deployment	Neon Database (serverless Postgres)

Architecture Overview

The application follows a modern React architecture with server components and client components clearly separated:

```
JustScore
├── app/                    # Next.js 14 app directory
│   ├── api/                # API routes
│   ├── dashboard/          # Dashboard pages and components
│   │   ├── components/     # Dashboard-specific components
│   │   ├── onboarding/     # Onboarding flow
│   │   └── teams/          # Team management
├── assets/                 # Static assets
├── components/             # Shared component library
│   ├── ui/                 # UI components
│   └── logo/               # Branding components
├── hooks/                  # Custom React hooks
├── lib/                    # Utility functions and shared code
│   ├── api/                # API client and utilities
│   ├── prisma.ts           # Prisma client setup
│   └── types/              # TypeScript type definitions
├── store/                  # Zustand stores for state management
└── styles/                 # Global styles
```

Database Schema

The application uses Prisma ORM with PostgreSQL. The schema is defined in `schema.prisma` and contains the following main entities:

Core Entities:

- **AppUser**: Store user authentication and profile information
- **GTeam**: Teams that contain members and activities
- **TeamMember**: User membership within a team
- **TeamFunction**: Function categories for teams
- **JobGrade**: Levels/grades for team members
- **Action/ActionCategory**: Templates for business actions/activities
- **OrgAction**: Concrete actions assigned to teams
- **MemberScore**: Performance scores for team members on actions
- **StructuredFeedback**: Structured feedback for team members
- **PerformanceReview**: Quarterly/annual performance reviews
- **AuditLog**: System audit logs for changes

Key Relationships:

- Users can own multiple teams and be members of multiple teams
- Teams have members and business actions
- Members receive scores, feedback, and reviews
- Business actions are categorized and scored

State Management

The application uses a combination of Zustand stores and TanStack Query (React Query) for state management:

Zustand Stores:

Located in the `/store` directory, they handle UI state and some cached data:

- **useConfigStore**: Configuration storage for onboarding process
- **useActivitiesStore**: Managing business activities state
- **useTeamStore**: Team management UI state
- **useMemberStore**: Team member management UI state
- **usePerformersStore**: Performance data visualization state
- **useProfileStore**: User profile state
- **useReviewStore**: Performance review management state

- **useOnboardingStore**: Onboarding flow state

React Query:

Used for server state management with optimistic updates, caching, and refetching strategies:

- API data fetching with automatic caching
- Mutations for data updates with optimistic UI updates
- Invalidation of queries on data changes

Authentication & Authorization

The application uses Clerk for authentication:

- User authentication via Clerk
- Authorization middleware in `middleware.ts`
- Server-side auth checks in API routes
- User synchronization between Clerk and the internal database

Key Components

UI Components

The application uses a combination of shadcn/ui components and custom components:

- **Core UI Components**: Located in `/components/ui/core`, these are base UI components like Button, Card, Dialog, etc.
- **Composite Components**: Located in `/components/ui/composite`, these are more complex components like ProfileCard, TeamCard, etc.
- **Layout Components**: Page layout components for consistent UI.

Business Logic Components

- **TeamPerformanceView**: Displays team performance metrics
- **PerformanceScoringModal**: Modal for scoring team member performance
- **MemberDashboard**: Individual member performance dashboard
- **GenerateReviewModal**: AI-powered performance review generation

API Structure

The application uses Next.js API routes with a RESTful design:

Main API Endpoints:

- **User Management**: `/api/user/*` - User profile, activities, teams

- **Teams**: `/api/teams/*` - Team CRUD operations
- **Members**: `/api/teams/:teamId/members/*` - Team member management
- **Activities**: `/api/business-activities/*` - Business activities
- **Performance**: `/api/teams/:teamId/members/:memberId/ratings` - Performance ratings

API Error Handling:

The application uses a centralized error handling system in `lib/api/error-handler.ts` that:

- Handles Prisma database errors
- Handles validation errors
- Provides consistent error responses

Onboarding Flow Architecture

The onboarding flow is a multi-step process guided by:

- **Middleware**: Route protection based on setup completion
- **Stores**: Configuration storage in the ConfigStore
- **Validation**: Step validation in useStepValidation hook
- **Navigation**: Smart navigation with the useOnboardingNavigation hook

Key Custom Hooks

The application uses custom hooks for business logic:

- **useTeamsManagement**: Team creation and management
- **useMemberManagement**: Member management
- **useActionsSelection**: Activity selection during onboarding
- **usePerformanceDistribution**: Analysis of performance distribution
- **useChartStyles**: Styling for performance charts

Reactivity and Events

The application uses a mix of React's built-in state management and custom event systems:

- **React State**: For local component state
- **Zustand**: For global application state
- **React Query**: For server state
- **Custom Events**: For cross-component communication (e.g., validation events)

Deployment and Environment

The application is configured to deploy to a production environment with:

- **Neon Database**: Serverless PostgreSQL via `@neondatabase/serverless`
- **Environment Variables**: DATABASE_URL, DIRECT_URL, etc.
- **Prisma Client**: Generated by Prisma ORM for database access

Performance Optimizations

- **Bundle Analyzer**: Configured with `@next/bundle-analyzer`
- **Dynamic Imports**: Used for code splitting
- **React.memo**: Used for heavy components
- **useCallback/useMemo**: Used for preventing unnecessary renders
- **Pagination**: Implemented for large data sets

Security Considerations

- **CSRF Protection**: Built into Next.js API routes
- **Data Validation**: Zod validation for user input
- **Auth Middleware**: Route protection via Clerk middleware
- **Data Access Control**: Team and member access checks

Error Handling Strategy

- **API Errors**: Centralized error handling in API routes
- **UI Error States**: Error boundaries and error states in components
- **Validation Errors**: Form validation errors with user feedback
- **Network Errors**: React Query error handling for network issues

Configuration Options

The application has the following configuration options:

- **Environment Variables**: DATABASE_URL, DIRECT_URL, etc.
- **Next.js Config**: Configured in `next.config.mjs`
- **Tailwind Config**: Configured in `tailwind.config.js`
- **Prisma Config**: Configured in `prisma/schema.prisma`

Development Tools and Workflows

- **TypeScript**: Static type checking
- **ESLint**: Code quality enforcement

- **Storybook**: Component development and testing
- **Prisma**: Database schema management and migrations

Third-Party Services Integration

- **Clerk**: Authentication and user management
- **Neon Database**: Serverless PostgreSQL database
- **OpenAI**: Used for generating AI-powered performance reviews

Future Extensibility Points

The architecture includes several points for future extension:

- **Subscription Tiers**: FREE, PREMIUM, ENTERPRISE
- **Team Function Customization**: Customizable team functions
- **Performance Review Templates**: Customizable review templates
- **Audit Logging**: Detailed audit logs for changes