TeamHub Mobile App Implementation

Version: 2.0

Date: October 06, 2025

Author: Joseph Hodgson

# Mobile

I'll create a comprehensive plan for converting TeamHub from a React web application to a React Native app using your Expo template. Let me first understand both codebases and then outline the conversion strategy.

Searched for "What is the current architecture of TeamHub? Show me the main frontend structure, routing setup, UI component libraries used, and how the client communicates with the backend API."

Planned conversion strategy (5 seconds)

Comprehensive Plan: Converting TeamHub to React Native with Expo

Based on my analysis of your current TeamHub architecture and React Native best practices, here's a detailed conversion plan:

📊 **Current Architecture Analysis**

**TeamHub Web Stack:**

* **Frontend**: React + TypeScript + Vite
* **Routing**: Wouter (lightweight routing)
* **UI**: shadcn/ui (Radix UI + Tailwind CSS)
* **State**: React Query + React Context (auth)
* **Backend**: Express.js + PostgreSQL (stays unchanged)
* **Auth**: Session-based + Replit Auth

🎯 **Conversion Strategy Overview**

**What Stays the Same:**

* ✅ Express backend API (no changes needed)
* ✅ PostgreSQL database and schema
* ✅ React Query for data fetching
* ✅ Business logic and API calls
* ✅ Authentication flow logic
* ✅ TypeScript types from @shared/schema.ts

**What Must Change:**

* ❌ All UI components (shadcn/ui → React Native components)
* ❌ Routing (wouter → Expo Router)
* ❌ Styling (Tailwind CSS → NativeWind or StyleSheet)
* ❌ HTML elements (div/span → View/Text)
* ❌ Platform APIs (web APIs → native equivalents)

📋 **Detailed Conversion Plan**

**Phase 1: Project Setup & Infrastructure** (Days 1-2)

1.1 Initialize Expo Project

* Create new Expo app with TypeScript template in your Expo repo
* Configure Expo Router for file-based navigation
* Set up folder structure matching TeamHub architecture

1.2 Shared Code Migration

* Copy shared/schema.ts (TypeScript types) - works as-is
* Copy API route types and interfaces
* Set up environment configuration for API endpoints

1.3 Dependencies Installation

# Core dependencies

- @tanstack/react-query (already compatible)

- expo-router (navigation)

- nativewind (Tailwind for React Native)

- @react-native-async-storage/async-storage (storage)

- react-native-safe-area-context (safe areas)

# UI Libraries (choose one approach):

- Option A: NativeWind (Tailwind-style, modern)

- Option B: React Native Paper (Material Design)

- Option C: Tamagui (cross-platform styled system)

**Phase 2: Navigation Architecture** (Days 3-4)

2.1 Expo Router Structure

Replace wouter routes with Expo Router file-based routing:

**Current wouter structure:**

/ → Landing

/register → Register

/login → Login

/dashboard → Dashboard (auth required)

/club → Club (auth required)

/team → Team (auth required)

/events → Events (auth required)

/dependents → Dependents (auth required)

/posts → Posts (auth required)

/settings → Settings (auth required)

**New Expo Router structure:**

app/

├── \_layout.tsx # Root layout with providers

├── (auth)/ # Auth group (public)

│ ├── \_layout.tsx

│ ├── index.tsx # Landing

│ ├── login.tsx

│ └── register.tsx

└── (app)/ # Main app group (protected)

├── \_layout.tsx # Tab navigation layout

├── (tabs)/

│ ├── \_layout.tsx # Bottom tabs

│ ├── index.tsx # Dashboard

│ ├── club.tsx

│ ├── team.tsx

│ ├── events.tsx

│ ├── dependents.tsx

│ ├── posts.tsx

│ └── settings.tsx

└── role-selection.tsx # Role wizard

2.2 Auth Protection

* Create middleware in \_layout.tsx to check authentication
* Redirect logic using Expo Router's useRouter()
* Session storage using AsyncStorage

**Phase 3: Core Components Conversion** (Days 5-8)

3.1 UI Component Mapping

**HTML → React Native Element Replacements:**

<div> → <View>

<span>, <p> → <Text>

<button> → <TouchableOpacity> or <Pressable>

<input> → <TextInput>

<img> → <Image>

<a> → <Link> (Expo Router) or <TouchableOpacity>

**shadcn/ui → React Native Component Mapping:**

Button → Custom TouchableOpacity with styling

Card → View with shadow/border styling

Input → TextInput with wrapper

Select → Custom Picker or bottom sheet

Dialog → Modal component

Toast → react-native-toast-message

Form → react-hook-form (works in RN)

Tabs → @react-navigation/material-top-tabs

3.2 Layout Components

* Create AppLayout equivalent with React Native navigation
* Implement sidebar → bottom tab navigation
* Create header component using React Navigation header
* Add safe area handling for iOS notch/Android navigation

3.3 Common Components to Build

src/components/

├── ui/

│ ├── Button.tsx # Custom button component

│ ├── Card.tsx # Card wrapper with shadow

│ ├── Input.tsx # Styled TextInput

│ ├── Avatar.tsx # Profile image component

│ ├── Badge.tsx # Status badges

│ └── EmptyState.tsx # No data state

├── layout/

│ ├── Screen.tsx # Screen wrapper with safe area

│ ├── Header.tsx # Navigation header

│ └── TabBar.tsx # Custom tab bar

└── forms/

├── FormField.tsx # Form field wrapper

└── FormError.tsx # Error display

**Phase 4: Styling System** (Days 9-10)

4.1 NativeWind Setup (Recommended)

* Configure NativeWind for Tailwind-like styling
* Create theme configuration matching current design
* Set up dark mode support

**Migration Example:**

// Web (Tailwind)

<div className="flex items-center justify-between p-4 bg-white rounded-lg shadow">

// React Native (NativeWind)

<View className="flex flex-row items-center justify-between p-4 bg-white rounded-lg shadow">

4.2 Theme System

src/theme/

├── colors.ts # Color palette

├── spacing.ts # Spacing scale

├── typography.ts # Font sizes & families

└── shadows.ts # Shadow styles

**Phase 5: Data Layer & API Integration** (Days 11-12)

5.1 React Query Setup

* Copy existing React Query setup
* Update API base URL configuration
* Handle network connectivity checks

**Platform-specific API URL:**

const API\_URL = Platform.select({

ios: 'http://localhost:5000', // iOS simulator

android: 'http://10.0.2.2:5000', // Android emulator

default: 'https://your-replit-url' // Production

});

5.2 Storage Migration

// Web

localStorage.setItem('token', value)

// React Native

import AsyncStorage from '@react-native-async-storage/async-storage'

await AsyncStorage.setItem('token', value)

5.3 Auth Hook Updates

* Update useAuth hook for AsyncStorage
* Handle session persistence
* Implement biometric auth (optional)

**Phase 6: Page-by-Page Conversion** (Days 13-18)

**Conversion Order (by priority):**

1. **Authentication Pages** (Days 13-14)
   * Landing page
   * Login page
   * Register page
   * Role selection wizard
2. **Dashboard** (Day 15)
   * Widget components
   * Stats displays
   * Upcoming events list
   * Match results
3. **Core Features** (Days 16-17)
   * Club page (association)
   * Team page (creation/management)
   * Events page (fixtures, availability)
   * Dependents page (player management)
4. **Secondary Features** (Day 18)
   * Posts page
   * Settings page
   * Profile management

**Phase 7: Platform-Specific Features** (Days 19-20)

7.1 Native Enhancements

// Add native features TeamHub could benefit from:

- expo-notifications # Push notifications for matches

- expo-calendar # Sync fixtures to device calendar

- expo-camera # Team/player photos

- expo-contacts # Share contact info

- expo-image-picker # Upload images

- expo-sharing # Share match results

7.2 Offline Support

* Implement offline data caching with React Query
* Add network status detection
* Queue mutations for when back online

**Phase 8: Testing & Refinement** (Days 21-23)

8.1 Testing Checklist

□ All routes navigate correctly

□ Forms submit and validate

□ API calls work (both online/offline)

□ Authentication flow complete

□ Role-based access control works

□ Images load properly

□ Lists scroll smoothly (FlatList optimization)

□ Safe areas handled (iOS notch, Android navigation)

□ Keyboard behavior correct (input focus, dismissal)

□ Platform-specific features tested

8.2 Performance Optimization

* Use FlatList for long lists (events, players)
* Implement pagination for large datasets
* Optimize images with expo-image
* Add loading skeletons

**Phase 9: Build & Deployment** (Days 24-25)

9.1 iOS Build

# Development build

eas build --platform ios --profile development

# Production build

eas build --platform ios --profile production

**Requirements:**

* Apple Developer account ($99/year)
* Mac for local testing (optional with EAS)

9.2 Android Build

# Development build

eas build --platform android --profile development

# Production build

eas build --platform android --profile production

**Requirements:**

* Google Play Developer account ($25 one-time)

9.3 Testing Distribution

* Use Expo Go for quick testing
* Create development builds for internal testing
* Set up TestFlight (iOS) and Internal Testing (Android)

🔄 **Backend Considerations**

Keep Backend Unchanged

Your Express backend can stay exactly as-is because:

* ✅ REST API works identically for mobile
* ✅ Session-based auth works (store session in AsyncStorage)
* ✅ CORS configuration may need mobile URLs added

Potential Backend Updates

// Add mobile app URL to CORS

app.use(cors({

origin: [

'http://localhost:8081', // Expo dev server

'exp://localhost:8081', // Expo scheme

// ... existing origins

],

credentials: true

}));

📊 **Timeline Summary**

| Phase | Duration | Complexity |
| --- | --- | --- |
| Setup & Infrastructure | 2 days | Low |
| Navigation | 2 days | Medium |
| Core Components | 4 days | High |
| Styling System | 2 days | Medium |
| Data Layer | 2 days | Low |
| Page Conversion | 6 days | High |
| Native Features | 2 days | Medium |
| Testing | 3 days | Medium |
| Build & Deploy | 2 days | Low |
| **Total** | **25 days** | **Mixed** |

💰 **Cost Breakdown**

**Required:**

* Apple Developer: $99/year
* Google Play: $25 one-time
* **Total**: ~$124

**Optional:**

* Expo EAS (build service): Free tier available, paid for production
* Physical test devices: $200-800
* Design assets/icons: $0-200

⚠️ **Key Challenges & Solutions**

| Challenge | Solution |
| --- | --- |
| shadcn/ui has no React Native version | Use NativeWind + build custom components |
| Complex form validation | react-hook-form works in RN, keep existing logic |
| Different navigation paradigms | Expo Router provides similar DX to wouter |
| Session management | AsyncStorage + React Query persistent cache |
| Platform differences (iOS/Android) | Platform.select() for conditional code |

🎯 **Recommended Approach**

**Option 1: Full React Native Conversion (Recommended for TeamHub)**

* **Pros**: Native performance, full access to device APIs, App Store presence
* **Cons**: More work upfront, maintain separate codebase
* **Best for**: TeamHub's feature set (calendar sync, notifications, camera)

**Option 2: Capacitor Wrapper (Alternative)**

* **Pros**: Faster, reuse existing React code
* **Cons**: WebView performance, limited native API access
* **Best for**: Simpler apps without heavy native features

📝 **Next Steps to Start**

1. **Set up Expo project** in your React Native repo
2. **Copy shared types** from TeamHub
3. **Build authentication flow** first (most critical)
4. **Create design system** (buttons, inputs, cards)
5. **Convert one page completely** as proof of concept (e.g., Dashboard)
6. **Iterate** through remaining pages

This plan gives you a production-ready React Native app while preserving your entire backend infrastructure. The Express API serves both web and mobile clients seamlessly.

Would you like me to start implementing this plan, beginning with Phase 1 (Project Setup)?