Table of Contents

# TokenMetrics Mobile App - Code Conventions

## 1. File Structure & Naming

### Directory Organization

src/  
├── assets/ # Static assets (images, fonts, etc.)  
├── components/ # Reusable UI components  
├── config/ # Configuration files  
├── enums/ # TypeScript enumerations  
├── hooks/ # Custom React hooks  
├── interfaces/ # TypeScript interfaces/types  
├── navigation/ # Navigation configuration  
├── screens/ # Screen components  
├── services/ # API and external services  
├── store/ # State management  
└── utils/ # Utility functions

### Naming Conventions

* **Directory & File Naming**:
  + Component Folders: camelCase
  + Component Files: PascalCase
  + Utility Folders & Files: camelCase
* components/button/Button.tsx  
  components/profileCard/ProfileCard.tsx  
  components/headers/Header.tsx  
  utils/formatDate.ts
* **Component Files**: Must match component name in PascalCase
* // Button.tsx  
  export const Button = () => { ... }
* **Type Files**: Group related types
* // types/user.ts  
  export type User = {  
   id: string;  
   name: string;  
  };  
    
  export type UserProfile = {  
   user: User;  
   settings: UserSettings;  
  };

## 2. Component Structure

### Component Organization

// Imports order  
import { useEffect, useState } from 'react';  
import { View, Text } from 'react-native';  
import { useAppStore } from '@/store/zustand/useAppStore';  
import { styles } from './styles';  
  
// Component definition  
export const ComponentName = () => {  
 // 1. Hooks  
 const [state, setState] = useState();  
 const { data } = useAppStore();  
  
 // 2. Derived state  
 const computedValue = useMemo(() => {  
 // ...  
 }, [dependencies]);  
  
 // 3. Effects  
 useEffect(() => {  
 // ...  
 }, [dependencies]);  
  
 // 4. Event handlers  
 const handlePress = () => {  
 // ...  
 };  
  
 // 5. Render methods  
 const renderItem = () => {  
 // ...  
 };  
  
 // 6. Return JSX  
 return (  
 <View>  
 {/\* JSX content \*/}  
 </View>  
 );  
};  
  
// Always add display name for better error stacks  
ComponentName.displayName = 'ComponentName';

### Component Display Names

* Always add display name to components for better error stacks
* Display name should match the component name
* Required for both regular and memo components

// Regular component  
export const Button = () => {  
 return <Pressable>...</Pressable>;  
};  
Button.displayName = 'Button';  
  
// Memo component  
export const Card = memo(() => {  
 return <View>...</View>;  
});  
Card.displayName = 'Card';  
  
// With props  
type HeaderProps = {  
 title: string;  
};  
export const Header = ({ title }: HeaderProps) => {  
 return <View>...</View>;  
};  
Header.displayName = 'Header';  
  
// HOC components  
export const withAuth = (WrappedComponent: ComponentType) => {  
 const WithAuth = (props: any) => {  
 return <WrappedComponent {...props} />;  
 };  
 WithAuth.displayName = `withAuth(${getDisplayName(WrappedComponent)})`;  
 return WithAuth;  
};

## 3. TypeScript Usage

### Type Definitions

* Use type for all type definitions (preferred)
* Use interface only when you need to take advantage of declaration merging
* Export types for reuse
* Use union types for finite sets of values

// Good  
type UserProps = {  
 name: string;  
 age: number;  
};  
  
type Status = "idle" | "loading" | "success" | "error";  
  
type StoreState = {  
 data: DataType;  
 isLoading: boolean;  
 setData: (data: DataType) => void;  
};  
  
// Only use interface when you need declaration merging  
interface Window {  
 myCustomProperty: string;  
}  
  
// Avoid  
interface UserProps {  
 name: string;  
 age: number;  
}

### Type Assertions

* Use type assertions sparingly
* Prefer type guards when possible

// Good  
if ("property" in object) {  
 // ...  
}  
  
// Avoid when possible  
const value = object as SomeType;

## 4. State Management

### Zustand Store Conventions

// Store definition  
type StoreState = {  
 data: DataType;  
 isLoading: boolean;  
 setData: (data: DataType) => void;  
};  
  
const useStore = create<StoreState>((set) => ({  
 data: initialData,  
 isLoading: false,  
 setData: (data) => set({ data }),  
}));

### Store Usage

* Use selective subscriptions
* Avoid storing derived state

// Good  
const data = useStore((state) => state.data);  
  
// Avoid  
const store = useStore();

## 5. Styling Conventions

### StyleSheet Usage

import { StyleSheet } from "react-native";  
  
const styles = StyleSheet.create({  
 container: {  
 flex: 1,  
 padding: 16,  
 },  
 // Group related styles  
 header: {  
 fontSize: 24,  
 fontWeight: "bold",  
 },  
});

### Style Organization

* Group related styles together
* Use consistent naming
* Avoid inline styles
* Use theme constants for colors, spacing, etc.

## 6. API Integration

### Service Structure

// services/api/endpoint.ts  
export const serviceFunction = async (  
 params: ParamsType,  
 authToken: string,  
): Promise<ResponseType> => {  
 const url = `${BASE\_URL}/endpoint`;  
 const controller = new AbortController();  
 const id = setTimeout(() => controller.abort(), DEFAULT\_TIMEOUT);  
  
 try {  
 const response = await fetch(url, {  
 method: "POST",  
 headers: {  
 "Content-Type": "application/json",  
 Authorization: `Bearer ${authToken}`,  
 },  
 body: JSON.stringify(params),  
 signal: controller.signal,  
 });  
  
 clearTimeout(id);  
  
 if (!response.ok) {  
 throw new Error("API Error");  
 }  
  
 return await response.json();  
 } catch (error) {  
 // Error handling  
 throw error;  
 }  
};

### API Service Types

type ServiceResponse<T> = {  
 data: T;  
 error: Error | null;  
};  
  
type UserServiceParams = {  
 id: string;  
 includeProfile?: boolean;  
};  
  
// Service function with type definitions  
const fetchUser = async (  
 params: UserServiceParams,  
): Promise<ServiceResponse<User>> => {  
 // Implementation  
};

## 7. Testing Conventions

### Test File Structure

describe("ComponentName", () => {  
 // Setup  
 beforeEach(() => {  
 // Common setup  
 });  
  
 // Test cases  
 it("should render correctly", () => {  
 // Test implementation  
 });  
  
 // Group related tests  
 describe("when user interacts", () => {  
 it("should handle interaction", () => {  
 // Test implementation  
 });  
 });  
});

## 8. Error Handling

### Error Boundary Usage

try {  
 // Risky operation  
} catch (error) {  
 if (error instanceof NetworkError) {  
 // Handle network errors  
 } else if (error instanceof ValidationError) {  
 // Handle validation errors  
 } else {  
 // Handle unknown errors  
 Sentry.captureException(error);  
 }  
}

## 9. Performance Optimization

### Memoization Usage

// Component memoization  
const MemoizedComponent = memo(Component, (prev, next) => {  
 return prev.id === next.id;  
});  
  
// Value memoization  
const memoizedValue = useMemo(() => {  
 return expensiveComputation(deps);  
}, [deps]);  
  
// Callback memoization  
const memoizedCallback = useCallback(() => {  
 handleAction(deps);  
}, [deps]);

## 10. Documentation

### Code Comments

/\*\*  
 \* Component description  
 \* @param {PropType} props - Props description  
 \* @returns {JSX.Element} Component description  
 \*/  
  
// Inline comments for complex logic  
const complexOperation = () => {  
 // Step 1: Initialize data  
 // Step 2: Process data  
 // Step 3: Return result  
};

### Type Documentation

/\*\*  
 \* Represents a user in the system  
 \* @type  
 \*/  
type User = {  
 /\*\* Unique identifier for the user \*/  
 id: string;  
 /\*\* User's full name \*/  
 name: string;  
 /\*\* User's email address \*/  
 email: string;  
};

## 11. Git Conventions

### Commit Messages

* Format: type(scope): message
* Types: feat, fix, docs, style, refactor, test, chore
* Example: feat(auth): add social login options

### Branch Naming

* Format: type/description
* Example: feature/social-login
* Example: bugfix/login-crash

These conventions should be followed to maintain consistency across the codebase and ensure maintainability. Always refer to this document when contributing to the project.