Table of Contents

# TokenMetrics Mobile App - Architectural Documentation

## 1. Technology Stack Overview

### Frontend Architecture

* **Framework**: React Native
* **State Management**: Zustand (Modern, lightweight state management)
* **Navigation**: React Navigation (Native Stack Navigator)
* **Language**: TypeScript
* **UI Components**: Custom components with React Native base components

### Directory Structure

src/  
├── assets/ # Static assets and resources  
├── 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

## 2. Core Architecture Components

### State Management (Zustand)

* Uses Zustand for state management with a clean, hooks-based API
* Main stores:
  + useAppStore: Manages core application state
    - Authentication state
    - User details
    - Navigation state
    - UI states
  + usePersistedChatStore: Handles chat-related persistent state

### Navigation Architecture

* Uses React Navigation’s Native Stack Navigator
* Structured navigation hierarchy:
  + Root Stack Navigator
    - Authentication flow (Sign In, Sign Up, etc.)
    - Main app flow (Protected routes)
    - Modal screens
* Implements drawer navigation for main app sections

### Authentication & User Management

* Comprehensive authentication flow
* User profile management
* Session validation and persistence
* Role-based access control (Basic/Premium users)

### UI/UX Architecture

* Component-based architecture with reusable elements
* Modular screen components
* Custom headers for different sections
* Modal system for overlays and popups
* Drawer-based navigation for main app sections

## 3. Key Features Implementation

### Chat System

* Prompt management
* Chat persistence
* Bot interaction states

### Profile Management

* User profile updates
* Password management
* Account settings
* Profile completion flow

### Security Features

* Secure authentication
* Password change functionality
* Session management
* Account deletion capability

## 4. Best Practices Implementation

### TypeScript Integration

* Strong typing throughout the application
* Interface definitions for API responses
* Enum usage for constants
* Type-safe navigation

### Code Organization

* Feature-based component organization
* Separation of concerns
* Reusable hooks and utilities
* Clear module boundaries

### State Management Principles

* Centralized state management
* Selective state subscription
* Persistent state handling
* Action encapsulation

## 5. Performance Considerations

### State Management Optimization

* Selective state updates
* Memoization of complex computations
* Efficient store subscriptions

### Navigation Performance

* Lazy loading of screens
* Optimized navigation options
* Proper screen unmounting

## 6. Security Measures

### Authentication Security

* Secure token management
* Session validation
* Protected routes
* Secure password handling

## 7. Testing Strategy

* Jest setup for unit testing
* Component testing capabilities
* Integration testing support

## 8. Analytics & Tracking

* PostHog integration for user analytics
  + User identification and tracking
  + Custom event tracking
  + User journey analysis
  + Feature usage metrics
* Event tracking implementation
  + Screen views
  + User interactions
  + Feature engagement
  + Error tracking

## 9. Error Handling & Monitoring

### Error Boundaries

* React Error Boundary implementation
* Fallback UI components
* Error recovery mechanisms
* Component-level error isolation

### Sentry Integration

* Real-time error tracking
* Performance monitoring
* Crash reporting
* Stack trace analysis
* Release tracking
* User context tracking
* Environment-specific monitoring

### Error Management

* Centralized error handling
* Custom error types
* User-friendly error messages
* Error logging and reporting
* Network error handling
* API error standardization

## 10. Future Considerations

### Scalability

* Modular architecture allows for easy feature addition
* Clear separation of concerns for maintainability
* Extensible state management pattern

### Maintenance

* Well-organized codebase for easy updates
* Clear documentation and type definitions
* Consistent coding patterns

## 11. Key Components Overview

### Chat Components

* ChatInput.tsx: Handles user input and message sending
* Prompts.tsx: Manages chat prompts and suggestions
* Markdown.tsx: Renders markdown content in chats

### Modal System

* BottomDrawer.tsx: Reusable bottom drawer component
* AppUpdateModal.tsx: Handles app update notifications
* DeleteAccountModal.tsx: Account deletion flow

### Profile & Settings

* ProfileAccountSection.tsx: User profile management
* DangerZoneSection.tsx: Critical account operations
* CompleteProfile.tsx: Profile completion workflow

### Authentication Screens

* SignIn.tsx: User login
* SignUpWithEmail.tsx: Email registration
* ChangePassword.tsx: Password management

## 12. API Requirements

### Authentication API

* Base URL Configuration: Configurable base URL for different environments
* Authentication Endpoints:
  + /auth/signup: Email-password registration with captcha
  + /auth/nftlogin: NFT-based authentication
  + OAuth Providers Integration:
    - Google OAuth
    - Twitter OAuth
    - Discord OAuth
    - Apple Sign-in
  + Session Management:
    - Token-based authentication
    - Refresh token handling
    - Session validation

### User Management API

* Profile Management:
  + /users/profile: GET user profile details
  + /users/updateUser: POST update user profile
  + /users/change-password: PUT password change
  + /users/: DELETE account management
* User Plan Management:
  + /userplan: POST user subscription plan details

### Chat System API

* Base URL: Separate chat bot base URL configuration
* Chat Endpoints:
  + /chat/message: POST send messages (deprecated)
  + /chat/save: POST save chat records (unused)
  + /chat/saveflag: GET save chat flags (unused)
* Streaming Chat:
  + Implements SSE (Server-Sent Events) for real-time chat
  + Handles chunked responses
  + Audio URL processing for voice responses

### Prompts Management API

* Example Prompts:
  + /chatbot/example-prompts: GET predefined prompts
* Saved Prompts:
  + /chatbot/saved-prompts: GET user’s saved prompts
  + /chatbot/saved-prompts: POST save new prompts
  + /chatbot/saved-prompts/{promptId}: DELETE remove saved prompts

### API Security Features

* Authorization header required for all authenticated endpoints
* Bearer token authentication
* Request timeout handling (DEFAULT\_TIMEOUT)
* CORS and content-type validation
* Error handling with detailed messages
* Abort controller implementation for request cancellation

### External Services Integration

* Supabase Authentication
* Kickbox Email Validation API
* Event Bus System for chat responses

### API Best Practices

* RESTful conventions
* Consistent error handling
* Request timeout management
* Response type validation
* Proper HTTP methods usage
* Structured response formats

## 13. Environment Configuration & Deployment Notes

### Environment Variables

* Production environment variables
* Staging environment variables
* Development environment variables
* Environment-specific configurations:
  + API endpoints
  + Feature flags
  + Service configurations
  + Analytics keys
  + Error tracking tokens

### Environment Files Structure

* .env: Main environment configuration
* expoEnv.ts: Production fallback configuration
  + Required for production releases
  + Mitigates Expo environment loading bug
  + Contains production-safe values
  + Should be kept in version control
  + Structure matches .env schema

### Important Deployment Notes

1. Production Release Requirements:
   * Must include expoEnv.ts fallback file
   * Required due to Expo bug with .env file loading in production builds
   * Fallback file should contain all necessary production configurations
2. Environment Variable Management:
   * Never commit .env files to version control
   * Keep expoEnv.ts updated with production values
   * Validate all required variables are present in both files
   * Document any changes to environment variables schema
3. Version Management:
   * Always update version numbers in both:
     + package.json: npm package version
     + app.json:
       - Expo app version and build number
       - Android versionCode (must be incremented for each Play Store submission)
       - iOS buildNumber
   * Ensure version numbers are synchronized
   * Follow semantic versioning (MAJOR.MINOR.PATCH)
   * Increment build number for each store submission
   * Document version changes in release notes
   * Version update checklist:
     1. Update version in package.json
     2. Update version in app.json
     3. Increment iOS buildNumber in app.json
     4. Increment Android versionCode in app.json
     5. Document changes in release notes
4. Build Process Considerations:
   * Verify environment configurations before builds
   * Test builds with fallback configuration
   * Validate sensitive information handling
   * Ensure proper variable loading in different environments

This architecture document reflects the current state of the TokenMetrics mobile app and provides a foundation for future development. The application follows modern React Native development practices with a focus on maintainability, performance, and user experience.