Frontend Technical Documentation

Overview

This documentation provides a comprehensive guide to the frontend architecture of our mobile financial application. The app is built using React Native with Expo, implementing a modern tech stack that includes TypeScript, NativeWind (TailwindCSS for React Native), and a custom UI component library based on Gluestack principles.

Tech Stack

- **Framework**: React Native with Expo
- **Language**: TypeScript
- **Styling**: NativeWind (TailwindCSS for React Native)
- **UI Library**: Custom Gluestack-based components
- **State Management**: Zustand
- **Navigation**: React Navigation
- **Internationalization**: i18n
- **Testing**: Pytest + Appium

Project Structure

Root-Level Configuration

File	Description
`App.tsx`	Entry point that initializes all providers (theme, safe area, gestures, i18n, navigation)
`app.json`	Expo configuration for app metadata, permissions, splash screen, plugins, and device support
`babel.config.js`	Babel setup with plugins for Gluestack, NativeWind, and module resolution
`eas.json`	Defines EAS (Expo Application Services) build configurations
`global.css`	TailwindCSS base layers (utilities, components)
`i18n.ts`	Language internationalization setup with async language detection (EN/FR)
`metro.config.js`	Metro bundler config integrated with NativeWind
`tailwind.config.js`	Extended TailwindCSS setup with Gluestack presets and custom color palette
`package.json`	Dependencies, scripts, and metadata for running, testing, and formatting
`tsconfig.json`	TypeScript configuration with strict typing and alias paths
`.npmrc`	Ensures compatibility across dependency trees with legacy-peer-deps

Component System

Our component architecture follows atomic design principles, with components organized by function and complexity.

Reusable Components (/components)

These building blocks are used throughout the app:

- **BackHeader** - Customizable top header with back button, icons, and title support
- **BitcoinPrice** - Displays live Bitcoin price with loading indicator
- **ButtonsTrain** - Styled tab-like button row for toggling between categories
- **BuyNSell** - Complete form with Buy/Sell logic, type selection, amount input, and validation
- **BuySellOrderBook** - Order book visualization for bid/ask markets
- **CandlestickChart** - SVG-based price candlestick charts
- **CryptoMarketCard** - Mini dashboard card showing real-time crypto data and sparkline
- **DatePicker** - Cross-platform date picker using native DateTime picker
- **DropdownTothor** - Stylized dropdown with slider integration and stateful logic

All components use:

- NativeWind for styling
- Box/Text primitives from the custom UI kit
- ThemeProvider for theming support

Auth Components (/components/auth)

These components manage user signups, verification, and input forms:

- **Signup Flow**: `SignupEmail`, `SignupVerifyEmail`, `SignupPhoneNumber`, `SignupVerifyPhone`, `SignupPersonalInformation`, `SignupAddress`, `SignupCreatePassword`
- **Supporting Components**: `MembershipModal`, `BackAuth`, `InputAuth`

These components work together to create a secure, step-based signup process with validation, biometric options, and progressive onboarding.

Identity Verification (/components/idVerification)

Used for KYC (Know Your Customer) processes:

- **CaptureID** - Capture user's ID photos
- **ConfirmQuality** - Allow users to confirm image quality
- **IdVerificationMain** - Orchestrates the flow between verification steps

Investments (/components/investments-page)

Components to display and manage investment options:

- **AvailableCards**, **OngoingCards**, **ExpiredCards** - Visually separate active, past, and potential investments
- **FullInvestmentInfo** - Detailed investment information

Market UI (/components/market)

Trading interface components:

- **CandleChart**, **DepthChart** - Visual market data representations
- **CoinOrderHistory**, **CoinInfo** - Coin-specific information displays

Profile UI (/components/profile)

- **CountryPhoneInput** - Combined country selector and phone input
- **ProfileHeader** - Displays profile image and user information

Savings (/components/savings)

- **ComputedInterests** - Shows interest calculations based on saving behavior

Settings (/components/settings)

- **ThemeSelector**, **LanguageSelector**, **CurrencySelector** - User preference controls fully integrated with ThemeProvider and i18n

Custom UI Library (/components/ui/)

Our atomic design system contains every button, slider, modal, etc., built from:

- Gluestack + NativeWind + Shadcn-like principles
- Reusable, platform-consistent design tokens

Components include:

- Basic UI elements: `button/`, `checkbox/`, `alert/`, `form-control/`
- **gluestack-ui-provider/** - Wraps the app with UI context and configuration

Screens and Navigation

Navigation Configuration

`screens/StackNavigator.tsx` defines the app's core navigation stack using React Navigation.

Major Screens

Screen	Purpose
`HomeScreen.tsx`	Landing screen with charts, portfolio summary, market trends
`PortfolioScreen.tsx`	Display holdings and value over time
`InvestmentsScreen.tsx`	Invest in cards and track returns
`TothorScreen.tsx`	Gamified experience or token-based features
`SimulatorScreen.tsx`	Simulates investment returns
`ChatScreen.tsx`	Chat interface or AI/Support assistant
`NotificationsScreen.tsx`	Lists in-app alerts
`TradingHistoryScreen.tsx`	Transaction logs and trades
`Layout.tsx`	Shared layout container for consistent look

Authentication Screens

Login and Signup screens using components from `/auth`.

Payment Screens

- `LinkYourCardScreen`
- `PaymentMethodScreen`
- `PaymentMethodAddedScreen`

PIN Code Screens

Complete PIN management flow:

- Create PIN
- Confirm PIN
- Enter PIN
- Change PIN

ID Verification Screens

- `FaceId.tsx`
- `TouchId.tsx`
- `VerifyIdentity.tsx`

Settings Screens

- `SettingsScreen.tsx`
- `PinSettingsScreen.tsx`

State Management

Located in the `/context` directory:

- **userStore.ts** - Zustand-based state for user data
- **pinStore.ts** - Local state for PIN management
- **dataStore.ts** - Shared context store

Utilities & APIs

API Layer

Path	Purpose
`utils/api/external/BinanceAPI.ts`	Fetch crypto market data from Binance
`utils/api/external/GoogleAPI.ts`	Autocomplete or address services
`utils/api/internal/sql/handleSQLite.ts`	Local DB integration for persistent data
`utils/api/internal/user/userApi.ts`	Endpoints for user-related operations

Utility Functions

- **crypto.ts** - Crypto-related calculations
- **passwordConvertor.ts** - Password encryption/decryption helpers
- **help.tsx** - Common helpers: formatting, value cleaning, etc.

Assets and Icons

- **utils/constants/Icons.tsx** - Exports SVG/TSX-based icons for coins, UI elements, etc.
- **assets/images/** - Holds image resources like splash, logo, icon, and other brand assets

Testing Suite

The `/tests` directory contains:

File	Purpose
`test_home_screen.py`	Validates rendering and interactivity on home screen
`test_tothor_screen.py`	Gamified feature flow tests
`test_card_flow.py`	Investment card interactions
`test_settings_page.py`	Settings toggles (theme, lang, currency)
`test_2fa_pin_settings.py`	PIN management and validation

Additionally, `test_screens/` contains specific screen-focused tests for Login, Market, Signup, and Portfolio.

The testing stack uses pytest + Appium for real device or simulator automation.

Internationalization

The i18n system includes:

- Translation files: `locales/en.json`, `locales/fr.json`
- Auto-language detection based on device locale or stored preference
- Global access via `useTranslation()` hook

Theming and Styling

- Theme values (light/dark) provided via ThemeProvider
- Color palette extended in `tailwind.config.js`
- Light/dark mode supported across all components
- Animations and shadows customized using Tailwind presets

Development Tooling

- **Prettier** with Tailwind plugin: `prettier.config.js`
- **Linting** via ESLint: `"eslintConfig": "universe/native"`
- **NativeWind** configuration in `tailwind.config.js`
- **Build profiles** in `eas.json` (e.g., preview without credentials)

Getting Started

Prerequisites

- Node.js (v16+)
- Yarn or npm
- Expo CLI
- iOS Simulator or Android Emulator (or physical device)

Installation

1. Clone the repository
2. Install dependencies:

bash

Copy

```
yarn install
```

3. Start the development server:

bash Copy

```
yarn start
```

Running Tests

bash Copy

```
pytest tests/
```

Best Practices

1. **Component Development**
   - Use the UI library components whenever possible
   - Follow the established pattern for new components
   - Ensure all components support theming
2. **State Management**
   - Use Zustand for global state
   - Keep component state local when possible
   - Use context for theme, i18n, and other app-wide configurations
3. **Styling**
   - Use NativeWind classes for styling
   - Follow the design system for consistency
   - Avoid inline styles
4. **Testing**
   - Write tests for all new features
   - Ensure tests run on both iOS and Android

Conclusion

This frontend architecture provides a scalable, maintainable foundation for our fintech mobile application. The modular component system, combined with robust state management and styling solutions, enables rapid development while maintaining consistency and quality.