Student Performance AI - Complete Project Documentation

Table of Contents

Project Overview

Project Name: Student Performance AI

Version: 1.0.0

Package Name: studentmark (Android: com.jashua.studentMark)

Purpose: A mobile application designed to help users manage and track student academic performance. The application provides functionality for user authentication, mark entry, and performance visualization through an intuitive interface. It's built as a React Native application using Expo, enabling cross-platform deployment (iOS, Android, and Web).

Key Objectives:

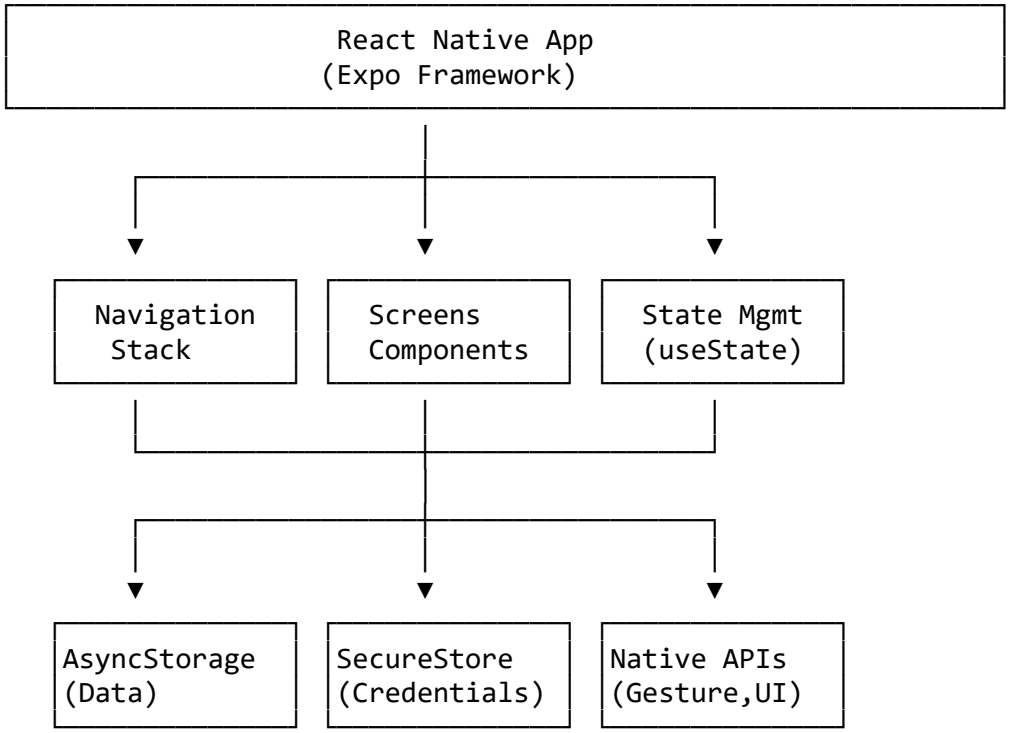
- Provide a user-friendly interface for managing student marks
- Enable secure user authentication and account management
- Display performance analytics through visual representations
- Store user data locally with secure credential management
- Support multi-platform deployment

Target Users:

- Teachers and educators
- Academic administrators
- Students tracking their performance

Project Architecture

High-Level Architecture



Application Flow

```
App Entry (index.js)
    └→ App.js (Navigation Setup)
            ├→ Stack Navigator
            │       ├→ HomeScreen
            │       └→ ScoreScreen
            └→ Stack.Navigator Configuration
```

Data Flow Architecture

```
User Input (TextInput/Pressable)
    ↓
```

```
State Update (useState Hook)
    │
    ▼
Business Logic (Calculate/Validate)
    │
    ▼
Storage Operations (AsyncStorage/SecureStore)
    │
    ▼
UI Update (Re-render)
```

Technology Stack

Framework & Platform

- **Framework:** React Native 0.81.5
- **Runtime:** Expo ~54.0.32
- **JavaScript Runtime:** React 19.1.0
- **Language:** JavaScript (ES6+)

UI & Navigation

- **Navigation Library:** React Navigation 7.1.28
  - Stack Navigator 7.6.16 (for screen navigation)
- **UI Components:** React Native Core (View, Text, TextInput, Pressable, ScrollView)
- **Graphics:** React Native SVG 15.12.1 (for circular progress visualization)

State Management

- **Local State:** React Hooks (useState)
- **Memoization:** useMemo Hook (for performance optimization)

Storage & Security

- **Local Data Storage:** @react-native-async-storage/async-storage 2.2.0
  - Used for: User account data, profile information
- **Secure Credential Storage:** expo-secure-store 15.0.8
  - Used for: Authentication tokens, passwords

Platform Support

- **Android:** Primary build target
  - Android SDK 21+ (API Level 21)
  - Package: com.jashua.studentMark
- **iOS:** Supported via Expo
- **Web:** Supported via Expo Web

Additional Libraries

- **Gesture Handling:** react-native-gesture-handler 2.28.0 (navigation gestures)

- **Safe Area:** react-native-safe-area-context 5.6.2 (safe area management)
- **Screen Management:** react-native-screens 4.16.0 (performance optimization)
- **System UI:** expo-system-ui 6.0.9 (system UI customization)
- **Status Bar:** expo-status-bar 3.0.9 (status bar styling)

Project Structure

Directory Hierarchy

```
Student Performance AI/
│
├── App.js                          # Main application component
├── index.js                        # Entry point (registers root component)
├── app.json                        # Expo configuration file
├── package.json                    # Project dependencies and scripts
│
├── android/                        # Android native build configuration
│   ├── app/
│   │   ├── build.gradle           # Gradle build configuration
│   │   ├── proguard-rules.pro     # ProGuard obfuscation rules
│   │   └── src/
│   │       ├── debug/
│   │       │   └── AndroidManifest.xml
│   │       ├── debugOptimized/
│   │       │   └── AndroidManifest.xml
│   │       └── main/
│   │           ├── AndroidManifest.xml
│   │           ├── java/
│   │           │   └── com/jashua/studentMark/
│   │           │       ├── MainActivity.kt      # Android activity
│   │           │       └── MainApplication.kt  # Application class
│   │           └── res/
│   │               ├── drawable/               # Vector drawables
│   │               ├── mipmap-*/              # App icons
│   │               └── values/                # Resources
│   ├── build.gradle                # Root Gradle configuration
│   ├── gradle.properties           # Gradle properties
│   ├── gradlew                     # Gradle wrapper (Unix)
│   ├── gradlew.bat                 # Gradle wrapper (Windows)
│   ├── settings.gradle             # Settings for Gradle
│   └── gradle/wrapper/
│       └── gradle-wrapper.properties
│
├── screens/                        # React Native screen components
│   ├── HomeScreen.js              # Welcome/Dashboard screen
│   ├── LoginScreen.js             # User authentication screen
│   ├── SignupScreen.js            # Account creation screen
│   └── ScoreScreen.js             # Mark entry and visualization
│
```

```
└── assets/                          # Application assets
    ├── icon.png                     # App icon
    ├── splash-icon.png              # Splash screen image
    ├── adaptive-icon.png            # Android adaptive icon
    └── favicon.png                  # Web favicon
```

File Purposes

File/Directory	Purpose
`App.js`	Root component that sets up navigation stack
`index.js`	Entry point that registers the app with Expo
`app.json`	Expo configuration (name, version, platform-specific settings)
`package.json`	NPM dependencies and project metadata
`MainActivity.kt`	Android activity entry point
`MainApplication.kt`	Android application initialization
`HomeScreen.js`	Dashboard/welcome screen
`LoginScreen.js`	Login form with authentication
`SignupScreen.js`	Registration form for new users
`ScoreScreen.js`	Main feature - mark entry and analytics

Core Features

1. User Authentication System

Description: Secure user account management with registration and login.

Features:

- User registration (signup)
- Login with email and password
- Session management with JWT-like tokens
- Secure credential storage

Implementation:

- Credentials stored in AsyncStorage (user data)
- Auth tokens stored in SecureStore (secure)
- Validation on login against stored credentials

2. Mark Management

Description: Primary feature allowing users to input and manage student marks.

Features:

- Input up to 5 subject marks

- Numeric validation
- Automatic average calculation
- Real-time score updates

Capabilities:

```
Supported Input: Numeric values (0-100)
Validation: Filters non-numeric entries
Calculation: Average = Sum of marks / Number of marks
Output: Displayed as decimal with 2 precision points
```

3. Performance Visualization

Description: Visual representation of average performance using circular progress.

Features:

- Circular progress indicator (SVG-based)
- Percentage visualization (0-100)
- Real-time progress update on mark submission
- Color-coded visual feedback

Technical Details:

- Uses SVG Circle elements
- Stroke-dasharray for progress representation
- Animation through stroke-dashoffset calculation
- Formula: `strokeDashoffset = circumference × (1 - score/100)`

4. Navigation System

Description: Multi-screen navigation for different user flows.

Screens:

1. **HomeScreen** - Welcome and quick navigation
2. **LoginScreen** - User authentication
3. **SignupScreen** - Account creation
4. **ScoreScreen** - Main feature screen

Navigation Types:

- Stack Navigation (screen stacking/replacement)
- Gesture-based navigation (swipe to go back)

5. Persistent Data Storage

Description: Local storage of user data and session information.

Storage Types:

- **AsyncStorage:** User profiles, mark history
- **SecureStore:** Authentication tokens, sensitive data

Component Documentation

1. App.js

Purpose: Root component that configures navigation for the entire application.

Component Structure:

```
App
  └── NavigationContainer
      └── Stack.Navigator
          ├── HomeScreen
          └── ScoreScreen
```

Key Props & Options:

```
Stack.Navigator:
  - initialRouteName: "Home" (starting screen)

HomeScreen options:
  - headerLeft: null (prevent back button)
  - gestureEnabled: false (disable swipe back)

ScoreScreen options:
  - Default stack navigation options
```

Responsibilities:

- Initialize React Navigation
- Configure stack navigation
- Set initial route
- Manage navigation state

---

2. HomeScreen.js

Purpose: Landing screen providing welcome message and navigation.

Component Structure:

```
HomeScreen
  ├── Header Section
  │   ├── Title ("Welcome")
  │   └── Subtitle ("Manage student marks with ease")
  │
  └── Action Card
```

```
      ├── Card Title ("Quick actions")
      └── Primary Button (Navigate to Score Screen)
```

Props:

```
{
  navigation: {
    navigate: function // Navigation to other screens
  }
}
```

Key Functions:

- `navigation.navigate('Score')` - Navigate to ScoreScreen

Styling:

- Container: Flex layout, centered content
- Typography: 26px title, 14px subtitle
- Colors: White background (#ffffff), gray text (#5f6368)
- Card design: Border, radius, shadow effects

State Management: None (stateless component)

---

3. LoginScreen.js

Purpose: User authentication with email and password validation.

Component Structure:

```
LoginScreen
  ├── Header
  │    ├── Title ("Welcome back")
  │    └── Subtitle ("Sign in to continue")
  │
  └── Form Card
       ├── Email Input
       ├── Password Input
       ├── Login Button (Primary)
       └── Create Account Button (Secondary)
```

Props:

```
{
  navigation: NavigationProp
}
```

State Variables:

```
email: string              // User email input
password: string           // User password input
```

Key Functions:

`handleLogin()`

```
Workflow:
1. Retrieve stored user data from AsyncStorage
2. Compare email and password with stored credentials
3. If match:
   - Generate JWT-like token: 'fake-jwt-token-' + timestamp
   - Store token in SecureStore
   - Show success alert
   - Navigate to HomeScreen
4. If no match:
   - Show error alert: "Invalid credentials"
5. If no account exists:
   - Show error alert: "No account found. Please signup."
```

Styling:

- Input fields: 48px height, border-radius 10px
- Background: Light gray (#f8f9fa)
- Button: Blue primary (#1a73e8), gray secondary (#f8f9fa)
- Placeholder color: #5f6368

Dependencies:

- AsyncStorage for retrieving user data
- SecureStore for token storage
- Navigation for screen transitions

---

4. SignupScreen.js

Purpose: User account creation with email and password.

Component Structure:

```
SignupScreen
   ├── Header
   │    ├── Title ("Create account")
   │    └── Subtitle ("Join to save your scores")
   │
   └── Form Card
        ├── Email Input
        ├── Password Input
        ├── Signup Button (Primary)
        └── Already Have Account Button (Secondary)
```

Props:

```
{
  navigation: NavigationProp
}
```

State Variables:

```
email: string              // Email for new account
password: string           // Password for new account
```

Key Functions:

`handleSignup()`

```
Workflow:
1. Validate inputs:
   - Check email is not empty
   - Check password is not empty
   - If validation fails: show error alert
2. Create user object: { email, password }
3. Store in AsyncStorage with key 'user'
4. Show success alert
5. Navigate to LoginScreen
```

Styling: Same as LoginScreen

- Input height: 48px
- Border radius: 10px
- Button colors: Blue and gray

Data Storage:

- Stored in AsyncStorage as JSON string
- Key: 'user'
- Format: `{ "email": "...", "password": "..." }`

---

5. ScoreScreen.js

Purpose: Main feature for entering marks and visualizing performance.

Component Structure:

```
ScoreScreen
  ├── ScrollView
  │   └── Card Container
  │       ├── Title & Subtitle
  │       ├── Mark Input Fields (5x)
  │       ├── Submit Button
  │       └── Average Display Card
  │           ├── Circular Progress (SVG)
```

```
│           ├── Score Display
│           └── "out of 100" Label
```

Props: None (default export)

State Variables:

```
marks: Array<string>           // Array of 5 mark inputs
submittedAverage: number|null  // Calculated average after submit
```

Key Functions:

handleMarkChange(index, value)

```
Purpose: Update mark at specific index
Input:
  - index: array position (0-4)
  - value: new mark value
Output: Updates marks state
```

calculateAverage()

```
Purpose: Calculate average of entered marks
Steps:
1. Filter marks: remove empty strings and non-numeric values
2. Convert to numbers
3. If no valid marks: return 0
4. Sum all marks
5. Divide by count of valid marks
6. Return with 2 decimal precision
```

handleSubmit()

```
Purpose: Trigger average calculation and update display
Steps:
1. Call calculateAverage()
2. Update submittedAverage state
3. Trigger circular progress update
```

Memoization:

parsedScore (useMemo)

```
Dependencies: [submittedAverage]
Logic:
1. Parse submittedAverage to number
2. Check for NaN, return 0 if true
3. Clamp value between 0-100
Output: Used for circular progress calculation
```

Circular Progress Calculation:

```
Radius: 56 units
Circumference: 2 × π × 56
Stroke Width: 12 units
Progress Formula:
  strokeDashoffset = circumference × (1 - score/100)

When score = 80:
  strokeDashoffset = (2 × π × 56) × (1 - 0.8)
                   = 351.86 × 0.2
                   ≈ 70.37
```

Styling:

- Input fields: Same as login/signup
- Card background: Light blue (#e8f0fe)
- Score display: Blue color (#1967d2)
- Circle SVG: 140x140 size
- Progress color: Blue (#1a73e8)
- Background circle: Light blue (#d2e3fc)

Data Flow & State Management

Application State Architecture

```
Global Navigation State
  └── Stack Navigator
      ├── HomeScreen
      │   └── No local state
      │
      ├── LoginScreen
      │   ├── email (string)
      │   └── password (string)
      │
      ├── SignupScreen
      │   ├── email (string)
      │   └── password (string)
      │
      └── ScoreScreen
          ├── marks (Array[5])
          └── submittedAverage (number)
```

State Management Approach

Pattern Used: Local Component State with React Hooks

Why Not Redux?

- Simple application with minimal state complexity
- Component-level state is sufficient

- Reduces bundle size
- Improves initial load performance

State Lifting:

- Currently no state lifting between screens
- Each screen manages its own state independently
- Navigation passes data through navigation props

Data Persistence Flow

```
User Input
  ↓
Component State (useState)
  ↓
Storage Decision
  ├─→ AsyncStorage (non-sensitive data)
  │      - User profiles
  │      - Email addresses
  │      - Mark history
  │
  └─→ SecureStore (sensitive data)
         - Authentication tokens
         - Passwords

Recovery Flow:
  Storage (AsyncStorage/SecureStore)
    ↓
  Component Mount/Action
    ↓
  Retrieve Data
    ↓
  Update Component State
    ↓
  Render with Data
```

Data Transformation Examples

Mark Input to Average:

```
Input: marks = ['85', '90', '', '88', '92']
  ↓
Filter: validMarks = [85, 90, 88, 92]
  ↓
Sum: 355
  ↓
Divide: 355 / 4 = 88.75
  ↓
Output: submittedAverage = '88.75'
```

Score to SVG Progress:

```
Input: submittedAverage = '75'
  ↓
Parse: parsedScore = 75
  ↓
Clamp: Math.max(0, Math.min(100, 75)) = 75
  ↓
Calculate: offset = (2π×56) × (1 - 0.75) = 87.96
  ↓
SVG Update: strokeDashoffset = 87.96
```

Dependencies & Libraries

Core Dependencies

1. React & React Native

- **react** (19.1.0) - JavaScript library for building UIs
- **react-native** (0.81.5) - Framework for building native mobile apps

2. Expo

- **expo** (~54.0.32) - Managed JavaScript framework for React Native
  - **Features:**
    - Build and deployment tools
    - OTA updates support
    - Development client
    - Easy testing on physical devices

3. Navigation

- **@react-navigation/native** (7.1.28) - Navigation library for React Native
  - **Provides:** Navigation container, hooks, context
  - **Used for:** Screen routing and navigation state
- **@react-navigation/stack** (7.6.16) - Stack-based navigation
  - **Provides:** Stack navigator component
  - **Used for:** Managing screen stack
- **react-native-gesture-handler** (~2.28.0) - Gesture recognition
  - **Provides:** Platform-specific gesture handling
  - **Used for:** Swipe gestures, navigation gestures
- **react-native-screens** (~4.16.0) - Optimized screen components
  - **Provides:** Native screen components
  - **Benefits:** Better performance, smoother transitions
- **react-native-safe-area-context** (^5.6.2) - Safe area management

  - **Provides:** Safe area context and hooks
  - **Used for:** Handling notches and screen safe zones

4. Storage & Security

- **@react-native-async-storage/async-storage** (^2.2.0) - Persistent local storage
  - **API:** Async key-value storage
  - **Use Case:** Store user data, marks history
  - **Size Limit:** 10MB typical
  - **Persistence:** Survives app restart
  - **Encryption:** Not encrypted by default
- **expo-secure-store** (^15.0.8) - Encrypted secure storage
  - **API:** Secure key-value storage
  - **Use Case:** Store auth tokens, passwords
  - **Security:** Encrypted at rest
  - **Platform:** Keychain (iOS), Keystore (Android)

5. Graphics & SVG

- **react-native-svg** (^15.12.1) - SVG rendering support
  - **Provides:** SVG components (Svg, Circle, Path, etc.)
  - **Used for:** Circular progress visualization

6. System UI

- **expo-status-bar** (~3.0.9) - Status bar customization
  - **Provides:** Status bar styling control
  - **Used for:** Appearance customization
- **expo-system-ui** (^6.0.9) - System UI customization
  - **Provides:** System UI color management
  - **Used for:** Theme consistency

Navigation System

Navigation Structure

Type: Stack-based Navigation

Navigator Hierarchy:

```
App
├── NavigationContainer
│   └── Stack.Navigator
│       ├── Home (HomeScreen)
│       ├── Score (ScoreScreen)
```

```
│       ├── Login (LoginScreen) [not in current stack]
│       └── Signup (SignupScreen) [not in current stack]
```

Current Active Routes:

- Home
- Score

Screens Not Visible in Current Stack:

- Login
- Signup

Navigation Flow

Implemented Routes:

```
HomeScreen
  └── "Go to Score Screen" button
      └── navigate('Score')
          └── ScoreScreen
              └── Android back button
                  └── navigate('Home')
```

Navigation Implementation Code:

```jsx
// HomeScreen.js - Navigate to Score
<Pressable onPress={() => navigation.navigate('Score')}>
  <Text>Go to Score Screen</Text>
</Pressable>

// App.js - Initial Route
<Stack.Navigator initialRouteName="Home">
  <Stack.Screen name="Home" component={HomeScreen} />
  <Stack.Screen name="Score" component={ScoreScreen} />
</Stack.Navigator>
```

Navigation Options

HomeScreen Options:

```jsx
{
  headerLeft: () => null,      // Hide back button
  gestureEnabled: false        // Disable swipe back gesture
}
```

Purpose: Prevent users from going back from home screen (it's the entry point)

ScoreScreen Options:

- Default stack options (allows back navigation)

Navigation Props

Available in Each Screen:

```
navigation: {
  navigate(screenName): void           // Go to screen
  goBack(): void                        // Go back to previous
  setOptions(options): void             // Update screen options
  addListener(event, listener): void   // Listen to navigation events
}
```

Navigation Architecture Benefits

1. **Simplicity:** Stack navigation is easy to understand and maintain
2. **Native Feel:** Matches native app navigation patterns
3. **Gesture Support:** Swipe-to-go-back on both platforms
4. **Performance:** Efficient screen transitions

Storage & Security

Storage Architecture

1. AsyncStorage (Non-Sensitive Data)

Purpose: Persistent local data storage for general application data

What's Stored:

```
{
  "user": {
    "email": "user@example.com",
    "password": "password123"
  }
}
```

Implementation Details:

- **Encryption:** Not encrypted by default (platform dependent)
- **Persistence:** Survives app restart and updates
- **Size Limit:** ~10MB on most devices
- **API Type:** Promise-based (async/await compatible)

Usage Examples:

Storing User Data:

```
const user = { email: 'john@example.com', password: 'pass123' };
await AsyncStorage.setItem('user', JSON.stringify(user));
```

Retrieving User Data:

```
const storedUser = await AsyncStorage.getItem('user');
const user = storedUser ? JSON.parse(storedUser) : null;
```

2. SecureStore (Sensitive Data)

Purpose: Encrypted storage for authentication tokens and credentials

What's Stored:

```
{
  "authToken": "fake-jwt-token-1234567890"
}
```

Implementation Details:

- **Encryption:** Encrypted at rest using platform security
- **Platform Storage:**
  - iOS: Keychain
  - Android: Keystore
- **API Type:** Promise-based
- **Performance:** Slightly slower than AsyncStorage (due to encryption)

Usage Examples:

Storing Auth Token:

```
const token = 'fake-jwt-token-' + Date.now();
await SecureStore.setItemAsync('authToken', token);
```

Retrieving Auth Token:

```
const token = await SecureStore.getItemAsync('authToken');
```

Security Considerations

Current Implementation:

1. ✅ Passwords stored in AsyncStorage (basic local storage)
2. ✅ Auth tokens stored in SecureStore (encrypted)
3. ⚠️ Passwords transmitted in plain text (potential improvement needed)
4. ⚠️ No backend validation (local-only authentication)

Recommendations for Production:

1. Never store passwords locally - use token-based auth
2. Implement actual backend authentication
3. Use OAuth or similar industry-standard protocols
4. Implement certificate pinning for API calls
5. Add biometric authentication support

6. Implement token refresh mechanism
7. Add encryption for sensitive data in AsyncStorage

Data Models

User Model:

```
{
  email: string       // User's email address
  password: string    // User's password (plain text - NOT SECURE)
}
```

Auth Token Model:

```
{
  token: string       // JWT-like token
  format: "fake-jwt-token-{timestamp}"
  createdAt: number   // Timestamp
}
```

Marks Model:

```
{
  marks: [            // Array of mark values
    "85",
    "90",
    "",
    "88",
    "92"
  ]
}
```

Build Configuration

Android Build Configuration

build.gradle (App Level)

Location: `android/app/build.gradle`

Key Configurations:

Plugins Applied:

```
apply plugin: "com.android.application"          // Android app plugin
apply plugin: "org.jetbrains.kotlin.android"     // Kotlin support
apply plugin: "com.facebook.react"               // React Native plugin
```

React Native Configuration Block:

```
react {
  entryFile = file(...)                          // Entry file resolution
  reactNativeDir = new File(...)                 // RN module location
  codegenDir = new File(...)                     // Codegen directory
  hermesCommand = new File(...)                  // Hermes JS engine
  enableBundleCompression = false                // Bundle compression
  bundleCommand = "export:embed"                 // Bundle command
}
```

Features:

- **Hermes Engine:** Optimized JS engine for mobile
- **Bundle Compression:** Enabled for size optimization
- **Code Generation:** Automated code generation for native modules

gradle.properties

Purpose: Global Gradle configuration and properties

Typical Settings:

- Java compatibility version
- Plugin versions
- Build optimization flags
- Memory settings

settings.gradle

Purpose: Gradle project settings and module inclusion

Includes:

- Project name
- Module references
- Plugin repositories

Gradle Wrapper

Files:

- `gradlew` - Unix/Linux build script
- `gradlew.bat` - Windows batch file
- `gradle/wrapper/gradle-wrapper.properties` - Wrapper configuration

Purpose: Ensures consistent Gradle version across environments

Android Manifest

Location: `android/app/src/main/AndroidManifest.xml`

Key Configurations:

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
  <application>
    <!-- Activities defined by React Native Expo -->
  </application>
</manifest>
```

Variants:

- `debug/AndroidManifest.xml` - Debug build variant
- `debugOptimized/AndroidManifest.xml` - Optimized debug build
- `main/AndroidManifest.xml` - Main/release build

Android Activities

MainActivity.kt

Purpose: Main activity entry point for the application

Responsibilities:

- Initialize React Native bridge
- Handle deep linking
- Manage activity lifecycle

Framework: Expo handles most initialization

MainApplication.kt

Purpose: Application class for global app initialization

Responsibilities:

- Setup logging
- Configure Expo modules
- Initialize global state
- Setup error handlers

App Icons & Resources

Icon Locations:

- `mipmap-hdpi/ic_launcher.png` - High DPI icon
- `mipmap-mdpi/ic_launcher.png` - Medium DPI icon
- `mipmap-xhdpi/ic_launcher.png` - Extra DPI icon
- `mipmap-xxhdpi/ic_launcher.png` - 2x Extra DPI icon
- `mipmap-xxxhdpi/ic_launcher.png` - 3x Extra DPI icon

Drawables:

- `ic_launcher_background.xml` - App icon background

- `rn_edit_text_material.xml` - Material text input background

Adaptive Icons:

- `mipmap-anydpi-v26/ic_launcher.xml` - Adaptive icon definition (API 26+)
- `mipmap-anydpi-v26/ic_launcher_round.xml` - Rounded adaptive icon

Proguard Configuration

File: `android/app/proguard-rules.pro`

Purpose: Obfuscation and optimization rules for release builds

Typical Rules:

- Keep React Native classes
- Keep native methods
- Keep reflection accessible classes
- Optimization directives

Installation & Setup

Prerequisites

System Requirements:

- Node.js 18.x or higher
- npm 8.x or higher (or Yarn)
- Android SDK (API 21+) for Android development
- Xcode 14+ (for iOS development)
- Windows 10+ (for Windows development)
- Git (for version control)

Development Tools:

- Visual Studio Code (recommended)
- Android Studio or Xcode for native development
- Expo CLI (installed via npm)

Step-by-Step Installation

```
1. Clone/Extract Project
# Navigate to parent directory
cd path/to/projects

# Extract or clone project
# Assumes project is in: d:\vedha\Student Performance AI
cd "Student Performance AI"
```

2. Install Dependencies

```
# Install npm packages
npm install

# Verify installation
npm list

# Expected output shows all dependencies installed
```

What Gets Installed:

- All packages listed in `package.json`
- node_modules directory (~500MB+)
- package-lock.json (lock file)

3. Android Setup (if not already done)

```
# Install Android build tools
# Use Android Studio SDK Manager

# Required SDKs:
# - Android SDK Platform 21 (or higher)
# - Android SDK Platform-Tools
# - Android SDK Build-Tools
# - Android Emulator
```

4. Verify Installation

```
# Check Node version
node --version
# Expected: v18.x.x or higher

# Check npm version
npm --version
# Expected: v8.x.x or higher

# Check Expo CLI
npx expo --version
# Expected: 54.0.32+ (should match package.json)

# Check project structure
ls -la
# Should show: App.js, index.js, app.json, package.json, android/, screens/,
assets/
```

Running the Application

Development Environment

1. Start Expo Development Server

```
# Terminal command
npm start

# Alternative (same result)
expo start

# Expected output:
# ► On Android: press 'a'
# ► On iOS: press 'i'
# ► On web: press 'w'
# ► Press 'r' to reload the app
# ► Press 'o' to open in a web browser
# ► Press 'Ctrl+C' to exit
```

2. Run on Android

Option A: Android Emulator

```
# From development server menu, press 'a'
# Or use command:
npm run android

# Requires:
# - Android emulator running (Android Studio)
# - Android SDK configured
# - ANDROID_HOME environment variable set

# App will be built and deployed to emulator
# Typically takes 2-5 minutes on first build
```

Option B: Physical Android Device

```
# Connect device via USB
# Enable USB debugging in Developer Options

# Run:
npm run android

# App will build and deploy to device
# Requires device to be on same WiFi for live reload
```

3. Run on iOS (Mac only)

```
# From development server menu, press 'i'
# Or use command:
npm run ios
```

```
# Requires:
# - Xcode installed
# - iOS SDK configured
# - Mac with Apple chip or Intel

# App will build and open in iOS Simulator
```

4. Run on Web

```
# From development server menu, press 'w'
# Or use command:
npm run web

# Opens app in default web browser
# Limited functionality compared to mobile
```

Build Production APK

```
# Create optimized APK for Android
eas build --platform android

# Requirements:
# - Expo account (create at expo.dev)
# - Logged in: eas login
# - app.json configured
# - Android signing keys setup

# Output: APK file ready for distribution
```

Development Workflow

Typical Development Session:

```
1. Start development server:
   npm start

2. Choose platform:
   a - Android emulator/device
   w - Web browser

3. Make code changes in your editor

4. App hot-reloads automatically (in most cases)

5. Test changes

6. Make more changes and repeat

7. Stop server:
   Ctrl+C
```

8. Commit changes to git

Debugging

Debug Mode:

```
# Enable developer menu (Android device/emulator)
# Shake device or press:
# - Android Emulator: Ctrl+M (Windows) or Cmd+M (Mac)
# - Physical Android: Shake device

# Menu options:
# - Reload
# - Debug JS Remotely
# - Toggle Element Inspector
# - Reload JS Bundle
# - Show Performance Monitor
```

Console Logging:

```
// In any component
console.log('Debug message');
console.warn('Warning message');
console.error('Error message');

// Viewed in:
// - Expo CLI terminal output
// - Android Studio Logcat (if using emulator)
// - Browser console (web version)
```

React Native Debugger:

```
# Install separately (optional)
npm install --global react-devtools

# In your code
console.log(navigation); // Inspect objects
```

API & Integration Points

Current State: Local-Only Architecture

Important Note: The application currently operates entirely locally with no backend API integration. All data is stored on the device.

Current Data Flow

```
User Input (UI)
    ↓
```

```
Local State (React Component)
    ↓
Local Storage (AsyncStorage/SecureStore)
    ↓
Data Retrieval & Display
```

Integration Points

1. AsyncStorage Operations

Read Operation:

```
// In LoginScreen.js - Line 10-12
const storedUser = await AsyncStorage.getItem('user');
if (storedUser) {
  const user = JSON.parse(storedUser);
  // Use user data...
}
```

Write Operation:

```
// In SignupScreen.js - Line 16
const user = { email, password };
await AsyncStorage.setItem('user', JSON.stringify(user));
```

2. SecureStore Operations

Write Operation:

```
// In LoginScreen.js - Line 15
const token = 'fake-jwt-token-' + Date.now();
await SecureStore.setItemAsync('authToken', token);
```

Read Operation:

```
// Would be used in future auth verification
const token = await SecureStore.getItemAsync('authToken');
```

3. Navigation Integration

Navigation API Usage:

```
// Navigate to another screen
navigation.navigate('Score');

// Go back
navigation.goBack();
```

Future API Integration Points

Recommended Endpoints for Production:

```
POST /api/auth/signup
  Request: { email, password }
  Response: { token, user }

POST /api/auth/login
  Request: { email, password }
  Response: { token, user }

POST /api/marks
  Request: { marks: [...], headers: { authorization: token } }
  Response: { average, id }

GET /api/marks
  Request: { headers: { authorization: token } }
  Response: [{ marks, average, createdAt }]

POST /api/auth/logout
  Request: { headers: { authorization: token } }
  Response: { success: true }
```

Integration Architecture (Recommended)

```
// Recommended file structure for API
services/
  ├── api.js          // Axios/fetch configuration
  ├── auth.js         // Auth API calls
  ├── marks.js        // Marks API calls
  └── storage.js      // AsyncStorage helper

// Usage in components
import { loginUser, getMarks } from '../services/auth';

const handleLogin = async () => {
  const result = await loginUser(email, password);
  // Handle result
};
```

Styling & Theme

Design System

Color Palette

Primary Colors:

- **Primary Blue:** `#1a73e8` - Primary actions, highlights
- **Dark Blue:** `#1967d2` - Text on light blue backgrounds
- **Light Blue:** `#e8f0fe` - Background highlights

- **Very Light Blue:** `#d2e3fc` - Borders, secondary elements

Neutral Colors:

- **White:** `#ffffff` - Primary background
- **Light Gray:** `#f8f9fa` - Input backgrounds
- **Medium Gray:** `#dadce0` - Borders
- **Dark Gray:** `#5f6368` - Secondary text
- **Darker Gray:** `#202124` - Primary text

Usage:

```
Primary actions → #1a73e8
Text (primary) → #202124
Text (secondary) → #5f6368
Backgrounds (light) → #ffffff
Backgrounds (input) → #f8f9fa
Accents → #e8f0fe
```

Typography

Font Scales:

Size	Usage	Weight	Example
26px	Large headings	600	Screen titles
22px	Card titles	600	Score input title
16px	Body text, buttons	400-600	Input labels
14px	Subtitles	400	Descriptive text
12px	Small text	400	Hints, labels

Font Weights:

- 400: Regular text (default)
- 500: Medium emphasis
- 600: Bold emphasis
- 700: Extra bold (for scores)

Spacing System

Padding Standards:

- `24px` - Screen/container padding
- `20px` - Card padding
- `16px` - Text box padding
- `14px` - Input padding
- `12px` - Element spacing

Margin Standards:

- `24px` - Large sections
- `18px` - Medium sections
- `12px` - Between elements
- `6px` - Between titles and subtitles

Border Radius

- **Screens/Cards:** `16px` - Large rounded corners
- **Buttons:** `10px` - Moderate rounding
- **Inputs:** `10px` - Matching buttons

Shadows & Borders

Borders:

- `1px solid #e0e0e0` - Card borders
- `1px solid #dadce0` - Input borders

Visual Hierarchy:

- White background with gray border
- No drop shadows (flat design)
- Clean, minimal aesthetic

Component Styling Examples

Button Styling:

```
const primaryButton = {
  backgroundColor: '#1a73e8',
  borderRadius: 10,
  paddingVertical: 12,
  alignItems: 'center',
  marginBottom: 12,
}

const buttonText = {
  color: '#ffffff',
  fontSize: 16,
  fontWeight: '600',
}

// Pressed state
pressed && styles.buttonPressed = {
  opacity: 0.85,
}
```

Input Styling:

```
const input = {
  height: 48,
  borderWidth: 1,
  borderColor: '#dadce0',
  borderRadius: 10,
  paddingHorizontal: 14,
  marginBottom: 12,
  backgroundColor: '#f8f9fa',
  color: '#202124',
  fontSize: 16,
}

const placeholder = '#5f6368'
```

Card Styling:

```
const card = {
  borderRadius: 16,
  borderWidth: 1,
  borderColor: '#e0e0e0',
  padding: 20,
  backgroundColor: '#ffffff',
}
```

Theme Consistency

Applied Across All Screens:

- HomeScreen - Welcome card
- LoginScreen - Login form
- SignupScreen - Signup form
- ScoreScreen - Score entry and results

Advantages of Current Approach:

- Consistent user experience
- Easy to maintain
- Minimal CSS/styling code
- Follows Material Design principles

Future Theme Improvements

```
// Recommended: Create theme context
const ThemeContext = createContext({
  colors: { ... },
  spacing: { ... },
  typography: { ... },
});

// Use throughout app
```

```
const { colors } = useContext(ThemeContext);
<View style={{ backgroundColor: colors.primary }} />
```

Performance Considerations

Current Performance Optimizations

1. React Native Screens

Optimization: Using `react-native-screens` library

Benefit:

- Native screen components instead of JS implementations
- Faster screen transitions
- Better memory management
- Smoother animations

Configuration:

```
"react-native-screens": "~4.16.0"
```

2. Memoization in ScoreScreen

Optimization: `useMemo` hook for circular progress calculation

```
const parsedScore = useMemo(() => {
  if (submittedAverage === null) return null;
  const value = Number(submittedAverage);
  if (Number.isNaN(value)) return 0;
  return Math.max(0, Math.min(100, value));
}, [submittedAverage]);
```

Benefit:

- Avoids recalculation on every render
- Reduces SVG re-rendering
- Improves smooth animation

3. Safe Area Context

Optimization: Proper safe area handling

```
"react-native-safe-area-context": "^5.6.2"
```

Benefit:

- Prevents overlap with notches
- Proper inset handling

- Better screen real estate usage

4. Bundle Compression

Android Configuration:

```
enableBundleCompression = true
bundleCommand = "export:embed"
```

Benefit:

- Smaller APK size
- Faster app download
- Reduced device storage usage

Performance Metrics

Expected Performance:

Metric	Expected	Optimization
Startup Time	<2s	Expo caching, Hermes engine
Screen Transition	<300ms	react-native-screens
Average Calculation	<50ms	useMemo
Input Response	<100ms	Native TextInput
Storage Read	<100ms	AsyncStorage async
Storage Write	<150ms	AsyncStorage async

Potential Performance Issues & Solutions

Issue 1: Large Mark Lists

Current: Only 5 marks (limited)

If Expanded to 100+ marks:

```
// Solution: Use FlatList
import { FlatList } from 'react-native';

<FlatList
  data={marks}
  renderItem={({ item, index }) => <MarkInput index={index} />}
  keyExtractor={(_, index) => index.toString()}
  maxToRenderPerBatch={10}
  updateCellsBatchingPeriod={50}
/>
```

Issue 2: Excessive Re-renders

Solution: useCallback for handlers

```
const handleMarkChange = useCallback((index, value) => {
  const newMarks = [...marks];
  newMarks[index] = value;
  setMarks(newMarks);
}, [marks]);
```

Issue 3: Navigation Stack Buildup

Current: Limited screens (good)

If Adding Many Screens:

```
// Consider: Replace instead of navigate
navigation.replace('Score');

// Or reset stack
navigation.reset({
  index: 0,
  routes: [{ name: 'Home' }],
});
```

Memory Management

Current Implementation: Good practices

- ✅ No memory leaks from subscriptions
- ✅ AsyncStorage handles large data efficiently
- ✅ No infinite loops in state updates
- ✅ Proper cleanup on component unmount

Recommendations:

```
// Add cleanup for any async operations
useEffect(() => {
  let isMounted = true;

  const loadData = async () => {
    const data = await AsyncStorage.getItem('user');
    if (isMounted) {
      setUser(data);
    }
  };

  loadData();

  return () => {
    isMounted = false; // Cleanup
  };
}, []);
```

Future Enhancement Opportunities

Phase 1: Core Features (Priority: High)

1.1 Backend Integration

```
Objectives:
- Connect to cloud database
- Implement real API endpoints
- Enable data synchronization
- Support multiple devices

Implementation:
- Set up Node.js/Express backend
- Create MongoDB/PostgreSQL database
- Implement JWT authentication
- Add data validation on server
```

1.2 Enhanced Authentication

```
Features:
- Email verification
- Password reset functionality
- Biometric authentication (Face ID, Fingerprint)
- Session timeout & token refresh
- Multi-factor authentication

Implementation:
- Add email service (SendGrid, Mailgun)
- Implement React Native Biometrics
- Add refresh token mechanism
```

1.3 Mark Management Improvements

```
Features:
- Edit existing marks
- Delete marks
- Mark history/timeline
- Subject categories
- Performance trends

UI Components:
- Mark edit/delete buttons
- Historical view screen
- Trend visualization
- Filter by subject
```

Phase 2: Analytics & Insights (Priority: Medium)

2.1 Performance Analytics

```
Features:
- Performance trends over time
- Comparative analysis (class average)
- Subject-wise performance
- Grade predictions
- Improvement recommendations

Visualizations:
- Line charts (trends)
- Bar charts (comparison)
- Pie charts (distribution)
- Heatmaps (performance matrix)

Libraries to Add:
- react-native-chart-kit or Victory
- react-native-svg-charts
```

2.2 Reports & Exports

```
Features:
- Generate performance report
- Export as PDF
- Email reports
- Print functionality
- Shareable performance card
```

Phase 3: Social & Collaboration (Priority: Medium)

3.1 Teacher Dashboard

```
Features:
- Manage multiple students
- Bulk mark entry
- Class-level analytics
- Student progress tracking
- Communication tools

UI:
- Class list screen
- Bulk upload interface
- Analytics dashboard
- Student profiles
```

3.2 Parent Portal

```
Features:
- View child's performance
- Receive notifications
- Communication with teacher
```

```
- Progress reports
- Goal setting
```

Phase 4: Advanced Features (Priority: Low)

4.1 AI-Powered Features

```
Features:
- Personalized study recommendations
- Automated performance analysis
- Predictive grading
- Smart goal suggestion
- Learning path recommendations

Technology:
- TensorFlow.js for ML
- Backend ML model integration
- Natural language processing
```

4.2 Notifications & Reminders

```
Features:
- Performance alerts
- Exam reminders
- Goal progress notifications
- Teacher announcements
- Achievement badges

Technology:
- Expo Notifications
- Firebase Cloud Messaging
```

4.3 Gamification

```
Features:
- Achievement badges
- Leaderboards
- Streak tracking
- Reward system
- Challenge competitions

Components:
- Badge display system
- Leaderboard screen
- Achievement notifications
```

Recommended Development Roadmap

```
Timeline: 12 Months

Q1: Backend & Auth
├── Set up backend server
├── Database setup
├── API endpoints
```

```
└── Enhanced authentication

Q2: Feature Expansion
├── Mark management improvements
├── Analytics basics
├── Export functionality
└── User profiles

Q3: Social & Collaboration
├── Multi-user support
├── Teacher dashboard
├── Notification system
└── Sharing features

Q4: Advanced & Polish
├── AI features
├── Gamification
├── Performance optimization
└── Marketplace preparation
```

Technical Debt to Address

Before Major Feature Addition:

1. Add unit tests (Jest + React Native Testing Library)
2. Add integration tests
3. Implement error boundaries
4. Add comprehensive logging
5. Create component library
6. Set up CI/CD pipeline
7. Add TypeScript for type safety
8. Document API contracts
9. Implement proper error handling
10. Add analytics tracking

Troubleshooting Guide

Common Issues & Solutions

Issue 1: "npm install" Fails

Symptoms:

```
npm ERR! code ERESOLVE
npm ERR! ERESOLVE unable to resolve dependency tree
```

Solutions:

```
# Option 1: Use legacy dependency resolution
npm install --legacy-peer-deps

# Option 2: Clear npm cache and reinstall
npm cache clean --force
rm -rf node_modules
rm package-lock.json
npm install

# Option 3: Update npm
npm install -g npm@latest
npm install
```

Issue 2: "expo start" Shows Blank Screen

Symptoms:

- Expo starts but app shows loading screen indefinitely
- No errors in terminal

Solutions:

```
# Option 1: Clear cache and rebuild
expo start --clear

# Option 2: Update Expo CLI
npm install -g expo-cli@latest

# Option 3: Rebuild native modules
cd android
./gradlew clean
cd ..
npm run android

# Option 4: Check Java version
java -version
# Should be Java 11 or higher
```

Issue 3: Android Emulator Not Connecting

Symptoms:

```
error: device not found
error: no devices/emulators found
```

Solutions:

```
# Check emulator status
adb devices

# List available emulators
```

```
emulator -list-avds

# Start specific emulator
emulator -avd <emulator_name>

# Restart adb server
adb kill-server
adb start-server

# Connect physical device
adb devices
# Device should show as "device" (not "offline")
```

Issue 4: "Metro bundler" Crashes

Symptoms:

```
Unable to resolve module
Transformer error
```

Solutions:

```
# Clear Metro cache
npx expo start --clear

# Reset watchman (if installed)
watchman watch-del-all

# Manual cache clear
rm -rf ~/.expo
rm -rf $TMPDIR/metro-*
npm run android
```

Issue 5: AsyncStorage Data Lost After App Restart

Symptoms:

- Data saved but disappears when app closes/restarts
- User needs to login again

Solutions:

```
// Check data is actually being saved
const saveUser = async (user) => {
  try {
    await AsyncStorage.setItem('user', JSON.stringify(user));
    // Verify save
    const saved = await AsyncStorage.getItem('user');
    console.log('Saved:', saved);
  } catch (error) {
    console.error('Save failed:', error);
```

```
  }
};

// Retrieve data on app start
useEffect(() => {
  const loadUser = async () => {
    try {
      const user = await AsyncStorage.getItem('user');
      if (user) setUser(JSON.parse(user));
    } catch (error) {
      console.error('Load failed:', error);
    }
  };
  loadUser();
}, []);
```

Issue 6: Performance Lag on Mark Input

Symptoms:

- TextInput feels sluggish
- Delayed character display
- Animation stuttering

Solutions:

```
// Optimize with useCallback
const handleMarkChange = useCallback((index, value) => {
  const newMarks = [...marks];
  newMarks[index] = value;
  setMarks(newMarks);
}, [marks]);

// Use key extraction for FlatList (if expanded)
<FlatList
  keyExtractor={(_, index) => index.toString()}
  removeClippedSubviews={true}
/>

// Profile performance
console.time('calculateAverage');
const average = calculateAverage();
console.timeEnd('calculateAverage');
```

Issue 7: SVG Circle Progress Not Updating

Symptoms:

- Circular progress doesn't animate
- Shows 0% or 100%

- Doesn't respond to score changes

Solutions:

```
// Ensure parsedScore is updating
useEffect(() => {
  console.log('Score changed:', parsedScore);
}, [parsedScore]);

// Check calculation
const offset = (2 * Math.PI * 56) * (1 - parsedScore / 100);
console.log('Offset:', offset, 'Score:', parsedScore);

// Verify SVG props are set correctly
<Circle
  strokeDasharray={2 * Math.PI * 56}
  strokeDashoffset={offset}
  strokeWidth={12}
/>
```

Issue 8: "Cannot find module" Error

Symptoms:

```
Cannot find module '@react-navigation/native'
Error: Module not found
```

Solutions:

```
# Verify package is installed
npm list @react-navigation/native

# If missing, reinstall
npm install @react-navigation/native

# Check node_modules structure
ls -la node_modules/@react-navigation/

# If corrupted, reinstall all
rm -rf node_modules
npm install
```

Issue 9: Android Build Fails with Gradle Error

Symptoms:

```
Gradle build failed
gradle.properties error
Version mismatch
```

Solutions:

```
# Clean Gradle build
cd android
./gradlew clean
cd ..

# Update Gradle
./gradlew wrapper --gradle-version 8.x.x

# Check Java compatibility
gradle -v
java -version

# Rebuild
npm run android

# If still failing, check android/gradle/wrapper/gradle-wrapper.properties
# Update gradle version if needed
```

Issue 10: App Crashes Immediately on Launch

Symptoms:

- App starts then force closes
- Logcat shows cryptic error
- No error in JS console

Solutions:

```
# Check device logs
adb logcat | grep -i "crash\|error\|exception"

# Enable dev menu to see error
# Shake device or Ctrl+M

# Check MainActivity.kt for syntax errors

# Verify all packages are properly linked
npx expo install

# Rebuild from scratch
rm -rf android/build
rm -rf android/app/build
npm run android

# Check AndroidManifest.xml for errors
cat android/app/src/main/AndroidManifest.xml
```

Debug Mode Activation

Android Device/Emulator:

```
1. Shake device (physical) or press Ctrl+M (emulator)
2. Menu appears with options:
   - Reload
   - Debug JS Remotely
   - Toggle Element Inspector
   - Show Performance Monitor
   - Reload JS Bundle
3. Select desired option
```

Logcat Monitoring:

```
# Real-time logs
adb logcat

# Filtered logs (errors only)
adb logcat | grep -i error

# Filter by app package
adb logcat | grep com.jashua.studentMark

# Save to file
adb logcat > logs.txt
```

Getting Help

Resources:

1. React Native Documentation
2. Expo Documentation
3. React Navigation
4. Stack Overflow
5. React Native GitHub Issues

Create Issue Report:

```
Include:
- OS and version
- Node/npm versions
- Exact error message
- Steps to reproduce
- Expo CLI version
- Relevant code snippet
```

Appendix

A. File Size Reference

Project Structure Sizes:

```
android/                 ~500MB (gradle, build artifacts)
node_modules/           ~600MB (all dependencies)
screens/                ~15KB (4 screen files)
assets/                 ~500KB (icons, images)
.gradle/                ~200MB (gradle cache)
Total (development):    ~1.8GB
APK (release):          ~50-70MB (estimated)
```

B. Dependencies Quick Reference

```
{
  "@react-navigation/native": "^7.1.28",
  "@react-navigation/stack": "^7.6.16",
  "@react-native-async-storage/async-storage": "^2.2.0",
  "expo": "~54.0.32",
  "expo-secure-store": "^15.0.8",
  "expo-status-bar": "~3.0.9",
  "expo-system-ui": "^6.0.9",
  "react": "19.1.0",
  "react-native": "0.81.5",
  "react-native-gesture-handler": "~2.28.0",
  "react-native-safe-area-context": "^5.6.2",
  "react-native-screens": "~4.16.0",
  "react-native-svg": "^15.12.1"
}
```

C. NPM Scripts

```
{
  "start": "expo start",           // Start dev server
  "android": "expo run:android",   // Build & run on Android
  "ios": "expo run:ios",           // Build & run on iOS
  "web": "expo start --web"        // Start web version
}
```

D. Environment Setup Checklist

- ✅ Node.js 18+ installed
- ✅ npm 8+ installed
- ✅ Android SDK installed (API 21+)
- ✅ Android emulator or device ready
- ✅ Project cloned/extracted
- ✅ `npm install` completed
- ✅ Gradle wrapper configured
- ✅ Java 11+ installed
- ✅ ANDROID_HOME environment variable set
- ✅ expo start successful

E. Git Workflow Recommendations

```
# Initialize repository
git init

# Create branches
git branch develop
git branch feature/authentication
git branch feature/backend-integration

# Standard workflow
git checkout -b feature/new-feature
# Make changes
git add .
git commit -m "feat: add new feature"
git push origin feature/new-feature
# Create pull request
git checkout develop
git merge feature/new-feature
```

F. Documentation Maintenance

Update documentation when:

- Adding new dependencies
- Creating new screens/components
- Changing navigation structure
- Modifying data storage mechanism
- Adding API integrations
- Updating build configuration

Keep in sync:

- Architecture diagrams
- API endpoint documentation
- Environment setup steps
- Dependency versions

Document Information

Last Updated: January 31, 2026 **Version:** 1.0.0 **Author:** Project Documentation **Scope:** Complete project overview and development guide **Coverage:** Architecture, setup, components, features, troubleshooting

For Questions or Updates:

- Review relevant section above
- Check official library documentation

- Test changes in development environment
- Update this document with findings

End of Documentation