Video Converter - React Native Android App Constitution

Preamble

This constitution serves as the foundational document governing the development, architecture, and maintenance of the Video Converter Android application. Built with React Native, this app leverages on-device processing power to provide offline video conversion capabilities while maintaining exceptional performance, user experience, and code quality standards.

Core Principles

I. Component-Driven Development (NON-NEGOTIABLE)

- **Atomic Design Philosophy**: Components must follow atomic design principles (atoms, molecules, organisms, templates, pages)
- **Single Responsibility**: Each component serves one clear purpose with a focused API
- **Reusability First**: All UI elements must be designed as reusable components with configurable props
- **Self-Containment**: Components must encapsulate their own logic, styles, and dependencies
- **Prop Interface Clarity**: Clear TypeScript interfaces for all component props with comprehensive documentation

II. Architectural Foundation

```
src/
├── components/           # Reusable UI components
│   ├── atoms/           # Basic building blocks (Button, Input, Icon)
│   ├── molecules/       # Component combinations (SearchBar, VideoCard)
│   ├── organisms/       # Complex components (VideoList, ConversionPanel)
│   └── templates/       # Page-level layouts
├── screens/             # Application screens
├── navigation/          # Navigation configuration and guards
├── services/           # Business logic and external integrations
│   ├── video/          # Video processing services
│   ├── storage/        # File system operations
│   └── analytics/      # Usage tracking and error reporting
├── hooks/              # Custom React hooks for shared logic
├── utils/              # Pure utility functions and helpers
├── constants/          # Application-wide constants and configurations
├── assets/            # Static resources (images, fonts, videos)
├── styles/            # Global styles, themes, and design tokens
├── types/             # TypeScript type definitions and interfaces
├── context/           # React Context providers for global state
└── __tests__/         # Test files organized by feature
```

III. Design System & Visual Identity

- **Color Palette**: Strict adherence to the defined color system
  - Primary Blue: `#2f6690` - Main navigation and primary actions
  - Secondary Blue: `#3a7ca5` - Secondary buttons and highlights
  - Light Gray: `#d9dcd6` - Background surfaces and subtle elements
  - Dark Blue: `#16425b` - Headers, important text, and emphasis
  - Light Blue: `#81c3d7` - Progress indicators and success states
- **Typography Scale**: Consistent font sizes, weights, and line heights
- **Spacing System**: 8px base unit with consistent margins and padding
- **Component Variants**: Standardized component variations (size, color, state)

IV. Code Quality & Documentation Standards

- **Comprehensive Documentation**: JSDoc comments for all functions, classes, and complex logic
- **TypeScript Excellence**: Strict TypeScript configuration with no `any` types
- **Component Documentation**: Props interface documentation with examples
- **Inline Comments**: Explaining business logic, algorithms, and non-obvious code
- **Architecture Decision Records (ADRs)**: Document significant architectural choices
- **README Standards**: Detailed README files for each major module

V. Styling & Design Implementation

- **Tailwind CSS with NativeWind**: Primary styling solution for consistency
- **Design Tokens**: Centralized design system with semantic naming
- **Responsive Design**: Mobile-first approach with device-specific adaptations
- **Dark/Light Theme Support**: Built-in theme switching capability
- **Accessibility**: WCAG 2.1 AA compliance with proper contrast ratios
- **No Inline Styles**: All styling through Tailwind classes or StyleSheet objects

VI. JavaScript/TypeScript Excellence

- **Modern ES6+ Features**: Destructuring, template literals, async/await
- **Functional Programming**: Pure functions, immutable data patterns
- **Custom Hooks**: Reusable logic abstraction through custom hooks
- **Error Boundaries**: Comprehensive error handling and recovery
- **Performance Optimization**: Memoization, lazy loading, and efficient re-renders

Technology Stack Specifications

Core Technologies

- **React Native**: 0.73+ with New Architecture (Fabric/TurboModules)
- **TypeScript**: 5.0+ with strict configuration
- **NativeWind**: Tailwind CSS for React Native styling
- **FFmpeg Kit**: Advanced video processing capabilities
- **React Navigation v6**: Type-safe navigation with deep linking support
- **React Native File System (RNFS)**: File operations and management

- **React Native Document Picker**: File selection interface

Video Processing Stack

- **FFmpeg Kit React Native**: Core video conversion engine
- **React Native Background Job**: Background processing capabilities
- **React Native Progress**: Visual feedback for conversion progress
- **React Native FS**: File system operations and temporary file management

State Management Architecture

- **React Context API**: Global state management with typed contexts
- **Custom Hooks**: Encapsulated state logic for specific features
- **Zustand**: Lightweight state management for complex scenarios
- **React Query**: Server state management and caching (if applicable)

Development & Testing Tools

- **ESLint**: Code quality and consistency enforcement
- **Prettier**: Automated code formatting
- **Husky**: Git hooks for quality gates
- **Jest**: Unit testing framework
- **React Native Testing Library**: Component testing utilities
- **Flipper**: Debugging and performance monitoring

Performance & Security Standards

Video Processing Optimization

- **Chunked Processing**: Break large files into manageable chunks
- **Memory Management**: Efficient memory usage with proper cleanup
- **Background Processing**: Non-blocking UI during conversions
- **Progress Tracking**: Real-time progress updates with cancellation support
- **Quality Presets**: Predefined quality settings for optimal performance
- **Temporary File Cleanup**: Automatic cleanup of intermediate files

Android-Specific Requirements

- **Permission Management**: Runtime permission requests with user-friendly explanations
- **Background Limitations**: Compliance with Android background execution limits
- **Storage Access Framework**: Proper file access following Android guidelines
- **ProGuard/R8 Optimization**: Code obfuscation and size optimization for releases
- **Target SDK Compliance**: Stay current with Google Play requirements

Security Implementation

- **File Validation**: Input sanitization and format validation
- **Permission Principle**: Minimal required permissions
- **Secure Storage**: Sensitive data protection using Keychain/Keystore

- **Network Security**: Certificate pinning for any external communications
- **Code Obfuscation**: Release build optimization and protection

Performance Benchmarks

- **App Launch Time**: < 2 seconds cold start
- **Video Processing**: Real-time progress feedback
- **Memory Usage**: < 200MB baseline, efficient cleanup
- **Storage Efficiency**: Minimal temporary file footprint
- **Battery Impact**: Optimized processing algorithms

Development Workflow & Governance

Branch Strategy & Git Workflow

```
main/
├── develop/           # Integration branch for features
├── feature/           # Feature development branches
├── hotfix/           # Critical bug fixes
└── release/          # Release preparation branches
```

Code Quality Gates

- **Pre-commit Hooks**: Linting, formatting, and basic tests
- **Pull Request Requirements**: Code review, automated testing, documentation updates
- **Continuous Integration**: Automated testing on multiple Android versions
- **Manual Testing**: Physical device testing on various Android versions
- **Performance Testing**: Memory usage and processing speed validation

Testing Strategy

- **Unit Tests**: > 80% code coverage for utilities and services
- **Component Tests**: Visual regression testing for UI components
- **Integration Tests**: End-to-end video conversion workflows
- **Performance Tests**: Memory usage and processing speed benchmarks
- **Manual Testing**: User experience validation on physical devices

Documentation Requirements

- **Code Documentation**: Comprehensive JSDoc for all public APIs
- **Architecture Documentation**: System design and data flow diagrams
- **User Documentation**: Feature guides and troubleshooting
- **Developer Onboarding**: Setup guides and contribution guidelines

Quality Assurance Framework

Automated Quality Checks

- **TypeScript Compilation**: Zero compilation errors
- **ESLint Rules**: Custom rules for project-specific patterns
- **Prettier Formatting**: Consistent code formatting
- **Bundle Size Analysis**: Monitor and optimize app size
- **Performance Profiling**: Regular performance audits

Manual Quality Assurance

- **User Experience Testing**: Workflow validation and usability
- **Device Compatibility**: Testing across various Android devices
- **Edge Case Handling**: Error scenarios and recovery testing
- **Accessibility Testing**: Screen reader and navigation testing

Release Criteria

- All automated tests passing
- Manual QA approval
- Performance benchmarks met
- Security audit completed
- Documentation updated

Extensibility & Future-Proofing

Plugin Architecture

- **Modular Design**: Features as pluggable modules
- **Codec Support**: Extensible video format support
- **Filter System**: Pluggable video filters and effects
- **Export Options**: Configurable output formats and quality settings

Internationalization (i18n)

- **String Resources**: Externalized strings for easy translation
- **RTL Support**: Right-to-left language compatibility

Governance & Decision Making

Architecture Review Board

- **Technical Decisions**: Major architectural changes require team consensus
- **Performance Impact**: All changes must consider performance implications
- **Security Review**: Security-related changes require specialized review

Version Management

- **Semantic Versioning**: MAJOR.MINOR.PATCH versioning scheme
- **Release Notes**: Comprehensive change documentation

Document Version: 1.0.0
Ratification Date: September 17, 2025
Last Amendment: September 17, 2025
Next Review Date: December 17, 2025

This constitution is a living document that evolves with the project needs while maintaining core principles and quality standards. All amendments require team consensus and documentation of rationale.