Implementation Plan: Mobile Video Converter Android App

Branch: `001-mobile-video-converter` | **Date**: September 17, 2025 | **Spec**: ../001-mobile-video-converter/spec.md
Input: Feature specification from `/specs/001-mobile-video-converter/spec.md`

Execution Flow (/plan command scope)

```
1. Load feature spec from Input path
   → ☑ Loaded successfully from 001-mobile-video-converter/spec.md
2. Fill Technical Context (scan for NEEDS CLARIFICATION)
   → ☑ Project Type: Mobile (React Native Android app)
   → ☑ Structure Decision: Option 3 (Mobile + API structure)
3. Fill the Constitution Check section based on the content of the constitution document.
   → ☑ Constitution requirements loaded and evaluated
4. Evaluate Constitution Check section below
   → ☑ No violations detected in initial design approach
   → ☑ Progress Tracking: Initial Constitution Check
5. Execute Phase 0 → research.md
   → ☑ Research phase completed
6. Execute Phase 1 → contracts, data-model.md, quickstart.md, .github/copilot-instructions.md
   → ☑ Design artifacts generated
7. Re-evaluate Constitution Check section
   → ☑ Post-design constitution check passed
   → ☑ Progress Tracking: Post-Design Constitution Check
8. Plan Phase 2 → Describe task generation approach (DO NOT create tasks.md)
   → ☑ Task planning approach documented
9. STOP - Ready for /tasks command
```

IMPORTANT: The /plan command STOPS at step 8. Phases 2-4 are executed by other commands:

- Phase 2: /tasks command creates tasks.md
- Phase 3-4: Implementation execution (manual or via tools)

Summary

Mobile Video Converter is an offline Android application that converts video files to web-optimized MP4 format using device processing power. The app features a touch-optimized Material Design UI with real-time progress tracking, device resource management, and APK distribution capabilities. Technical approach leverages React Native with TypeScript, FFmpeg Kit for video processing, and component-driven architecture following atomic design principles.

Technical Context

Language/Version: React Native 0.73+ with TypeScript 5.0+ (strict configuration)
Primary Dependencies: FFmpeg Kit React Native, React Navigation v6, NativeWind (Tailwind CSS), React Native File System (RNFS)
Storage: Local device storage with file system operations (no external database)
Testing: Jest unit tests, React Native Testing Library, integration tests with real video files
Target Platform: Android 8.0+ (API level 26), ARM64 and ARM32 architectures
Project Type: Mobile - Single Android application with offline video processing
Performance Goals: App launch < 2 seconds, memory usage < 200MB baseline, efficient video conversion
Constraints: Offline operation only, device thermal management, battery optimization, APK distribution
Scale/Scope: Single-purpose mobile app, 3 core screens, comprehensive video format support

Constitution Check

GATE: Must pass before Phase 0 research. Re-check after Phase 1 design.

I. Component-Driven Development ☑

- ☑ React Native with atomic design principles (atoms/molecules/organisms/templates)
- ☑ TypeScript interfaces for all component props with comprehensive documentation
- ☑ NativeWind for Tailwind CSS styling (no inline styles)
- ☑ Self-contained, reusable components following single responsibility

II. Maximum Productivity and TypeScript Excellence ☑

- ☑ TypeScript 5.0+ strict configuration enforced
- ☑ Functional programming with custom hooks for reusable logic
- ☑ ES6+ features (destructuring, async/await, template literals)
- ☑ Zero `any` types allowed - strict type safety enforced

III. Test-Coverage ☑

- ☑ Jest unit tests and React Native Testing Library component tests
- ☑ Minimum 80% code coverage for utilities and services
- ☑ TDD approach: Tests written → User approved → Tests fail → Then implement
- ☑ Performance tests for video processing workflows

IV. Integration Testing ☑

- ☑ Video conversion end-to-end workflows testing planned
- ☑ File system operations testing with real video files
- ☑ Background processing and component interaction testing
- ☑ Test dataset snapshots for repeatable testing

V. Architecture & Performance Standards ☑

- ☑ Atomic design folder structure
- ☑ FFmpeg Kit integration with chunked processing and memory management

- ☑ Background processing with real-time progress tracking
- ☑ Performance targets: App launch < 2 seconds, memory < 200MB

Project Structure

Documentation (this feature)

```
specs/001-mobile-video-converter/
├── plan.md              # This file (/plan command output)
├── research.md          # Phase 0 output (/plan command)
├── data-model.md        # Phase 1 output (/plan command)
├── quickstart.md        # Phase 1 output (/plan command)
├── contracts/           # Phase 1 output (/plan command)
└── tasks.md             # Phase 2 output (/tasks command - NOT created by
/plan)
```

Source Code (repository root)

```
# Option 3: Mobile Application Structure
android/
├── app/
│   ├── src/
│   │   ├── main/
│   │   │   ├── java/
│   │   │   ├── res/
│   │   │   └── AndroidManifest.xml
│   │   └── test/
│   ├── build.gradle
│   └── proguard-rules.pro
├── gradle/
└── settings.gradle

src/
├── components/
│   ├── atoms/           # Basic building blocks
│   ├── molecules/       # Simple combinations
│   ├── organisms/       # Complex components
│   └── templates/       # Layout components
├── screens/
│   ├── MainScreen/
│   ├── SettingsScreen/
│   └── ResultsScreen/
├── services/
│   ├── VideoProcessor/
│   ├── FileManager/
│   └── DeviceMonitor/
├── hooks/
├── utils/
├── types/
```

```
└── navigation/

tests/
├── __mocks__/
├── components/
├── services/
├── integration/
└── performance/
```

Structure Decision: Option 3 (Mobile Application) - React Native Android app with offline video processing capabilities

Phase 0: Outline & Research

Research Tasks Completed:

1. **FFmpeg Kit React Native Integration**
   - Decision: Use FFmpeg Kit React Native for video processing
   - Rationale: Provides comprehensive video processing with mobile optimization, supports hardware acceleration
   - Alternatives considered: Native Android MediaMetadataRetriever (limited formats), ExoPlayer (playback focused)

2. **Device Resource Management Patterns**
   - Decision: Implement thermal monitoring with React Native Device Info + native thermal throttling
   - Rationale: Prevents device overheating during intensive video processing
   - Alternatives considered: No monitoring (risky), basic battery monitoring only (insufficient)

3. **Background Processing Architecture**
   - Decision: Use React Native Background Job with foreground service notifications
   - Rationale: Ensures conversion continues when app is minimized, provides user feedback
   - Alternatives considered: Foreground only (poor UX), WorkManager (complex setup)

4. **File System Operations**
   - Decision: React Native File System (RNFS) for cross-platform file operations
   - Rationale: Comprehensive file system API with Android Media Store integration
   - Alternatives considered: Native modules only (platform-specific), Expo FileSystem (limited capabilities)

5. **State Management for Conversion Progress**
   - Decision: Zustand for lightweight state management with persistence
   - Rationale: Simple, TypeScript-friendly, perfect for single-purpose app state
   - Alternatives considered: Redux (overkill), React Context (performance concerns), AsyncStorage only (no reactive updates)

Output: ☑ research.md completed with all technical unknowns resolved

Phase 1: Design & Contracts

Data Model Entities:

1. **VideoFile**: Source and converted video files with metadata
2. **ConversionJob**: Active/completed conversion processes with progress tracking
3. **AppSettings**: User preferences and configuration
4. **DeviceResources**: System monitoring data

API Contracts:

Since this is an offline mobile app, "contracts" refer to internal service interfaces:

- **VideoProcessorService**: Video conversion operations
- **FileManagerService**: File system operations
- **DeviceMonitorService**: Resource monitoring
- **SettingsService**: User preferences management

Test Scenarios:

- End-to-end video conversion workflow
- Device resource management during processing
- File system operations and storage management
- UI state management during conversion

Output: ☑ data-model.md, /contracts/*, quickstart.md, .github/copilot-instructions.md completed

Phase 2: Task Planning Approach

This section describes what the /tasks command will do - DO NOT execute during /plan

Task Generation Strategy:

- Load `.specify/templates/tasks-template.md` as base
- Generate tasks from Phase 1 design docs (service contracts, data model, quickstart)
- Each service interface → contract test task [P]
- Each entity → TypeScript model creation task [P]
- Each user story → integration test task
- Component implementation tasks following atomic design hierarchy
- Implementation tasks to make tests pass

Ordering Strategy:

- TDD order: Tests before implementation
- Dependency order: Types/Models → Services → Components → Screens → Navigation
- Atomic design order: Atoms → Molecules → Organisms → Templates → Screens
- Mark [P] for parallel execution (independent files)

Estimated Output: 35-40 numbered, ordered tasks in tasks.md covering:

- Project setup and configuration (5 tasks)
- Data models and types (5 tasks)
- Service layer implementation (8 tasks)
- Component library (12 tasks)
- Screen implementations (6 tasks)
- Integration and testing (5 tasks)
- APK build and distribution (4 tasks)

IMPORTANT: This phase is executed by the /tasks command, NOT by /plan

Phase 3+: Future Implementation

These phases are beyond the scope of the /plan command

Phase 3: Task execution (/tasks command creates tasks.md)
Phase 4: Implementation (execute tasks.md following constitutional principles)
Phase 5: Validation (run tests, execute quickstart.md, performance validation)

Complexity Tracking

No constitutional violations detected - all requirements align with established principles

No entries required - the mobile video converter implementation follows all constitutional requirements without needing complexity justifications.

Progress Tracking

This checklist is updated during execution flow

Phase Status:

- [x] Phase 0: Research complete (/plan command)
- [x] Phase 1: Design complete (/plan command)
- [x] Phase 2: Task planning complete (/plan command - describe approach only)
- [ ] Phase 3: Tasks generated (/tasks command)
- [ ] Phase 4: Implementation complete
- [ ] Phase 5: Validation passed

Gate Status:

- [x] Initial Constitution Check: PASS
- [x] Post-Design Constitution Check: PASS
- [x] All NEEDS CLARIFICATION resolved
- [x] Complexity deviations documented (none required)

Based on Constitution v1.0.0 - See `/.specify/memory/constitution.md`