Alarmise Android Development Prompts - Step by Step

Phase 1: Project Setup & Foundation

Prompt 1: Repository Initialization

First, thoroughly read and analyze the alarmise_context.md file in the project directory to understand the complete project requirements, core concept, user flow, and critical behavioral requirements.

Create a new Android project setup for "Alarmise" - a specialized alarm app based on the specifications in the md file. Initialize with:
- Modern Android architecture (MVVM pattern)
- Kotlin as primary language
- Minimum SDK 24 (Android 7.0) for broad compatibility
- Target SDK 34 (Android 14)
- Include essential dependencies for audio playback, background services, and UI components
- Set up proper gitignore for Android projects
- Create initial project structure with proper package naming (com.alarmise.app)
- Include README with basic project description

Prompt 2: Core Architecture Setup

Reference the alarmise_context.md file to ensure alignment with project requirements.

Set up the core architecture for Alarmise with these components:
- Create MainActivity with basic UI structure
- Set up MVVM architecture with ViewModel and Repository patterns
- Create data models for Alarm (startTime, endTime, isActive, mathPuzzle)
- Set up Room database for alarm storage
- Create AlarmRepository for data operations
- Add dependency injection setup (Hilt/Dagger)
- Include proper lifecycle management components

Phase 2: Basic UI Implementation

Prompt 3: Main UI Screen

Create the main alarm configuration screen for Alarmise with:
- Time picker for alarm start time (12/24 hour format support)
- Time picker for alarm end time with validation (must be after start time)
- Clear visual indicators showing alarm duration
- "Set Alarm" button with proper state management
- Display current active alarm status if any
- Use Material Design 3 components

- Implement proper input validation and error messages
- Add accessibility features for time pickers

Prompt 4: Alarm Status & Management UI

Create alarm management interface with:
- Active alarm display showing countdown to start time
- Visual progress indicator for alarm duration window
- Current status messages (waiting, active, dismissed)
- Emergency stop explanation (only via math puzzle)
- Next alarm schedule display
- Alarm history view (last few alarms and their outcomes)
- Settings screen for basic app preferences

Phase 3: Core Alarm Logic

Prompt 5: Alarm Scheduling System

Before implementing, review the alarmise_context.md file to understand the precise timing requirements and user flow specifications.

Implement the alarm scheduling system with:
- AlarmManager integration for precise timing
- Handle device reboot scenarios (BOOT_COMPLETED receiver)
- Create AlarmScheduler class with start/end time management
- Implement alarm state management (scheduled, active, dismissed, expired)
- Add proper timezone handling
- Include alarm persistence across app restarts
- Handle edge cases (same day vs next day alarms)
- Add logging for debugging alarm scheduling issues

Prompt 6: Background Service Architecture

Refer to alarmise_context.md file to understand the critical requirement for persistent playback across all app states (foreground/background/closed).

Create the background service for persistent alarm playback:
- Foreground service for alarm playback (Android 8.0+ compatibility)
- Service lifecycle management (start, stop, restart scenarios)
- Proper service binding between UI and background service
- Handle service destruction and recreation
- Implement service communication via LocalBroadcastManager
- Add proper notification channel for foreground service

- Include battery optimization handling instructions
- Service auto-restart mechanisms for system kills

Phase 4: Audio System Implementation

Prompt 7: Audio Playback Engine

Review the alarmise_context.md file to understand the non-negotiable requirement for continuous, non-stop alarm playback that cannot be bypassed.

Implement robust audio playback system with:
- MediaPlayer setup for continuous, looped alarm sound
- Audio focus management (handle calls, notifications)
- Volume control bypass (play at system alarm volume)
- Multiple fallback alarm sounds in case of file issues
- Handle audio hardware changes (headphones disconnect)
- Implement fade-in alarm start (optional gentle wake-up)
- Audio session management for persistent playback
- Handle audio interruptions gracefully

Prompt 8: Audio Persistence & Recovery

Check alarmise_context.md for the critical behavioral requirement that audio must continue regardless of app state.

Ensure audio continues across all app states:
- Background playback when app is minimized
- Audio continuation when app is force-closed
- Playback recovery after system resource pressure
- Handle low-memory situations gracefully
- Audio stream type configuration (ALARM stream)
- Wake lock management for continuous playback
- Handle device sleep modes and doze optimization
- Audio routing management (speaker enforcement)

Phase 5: Math Puzzle System

Prompt 9: Math Puzzle Generator

Consult alarmise_context.md to understand that math puzzle solving is the ONLY method to dismiss the alarm - no alternative dismissal options should exist.

Create the math puzzle dismissal system:
- Random arithmetic problem generator (addition, subtraction, multiplication)
- Difficulty scaling based on time of day (easier in early morning)

- Puzzle display UI with large, clear numbers
- Input validation and answer checking
- Multiple attempts handling (3 wrong attempts = new puzzle)
- Puzzle timeout mechanism (auto-regenerate after 2 minutes)
- Accessibility features for puzzle solving
- Analytics for puzzle difficulty effectiveness

Prompt 10: Puzzle Integration & Flow

Reference alarmise_context.md to ensure the puzzle is properly integrated as the sole dismissal mechanism with proper cognitive engagement requirements.

Integrate math puzzle with alarm dismissal:
- Puzzle overlay screen that appears during active alarm
- Screen wake-up and unlock when alarm triggers
- Prevent puzzle bypass (disable back button, home button during puzzle)
- Correct answer processing and alarm dismissal
- Smooth transition from alarm to dismissal
- Handle puzzle solving edge cases (app crash during puzzle)
- Puzzle completion celebration/feedback
- Integration with alarm statistics

Phase 6: Critical System Features

Prompt 11: Auto-Stop Mechanism

Review alarmise_context.md to understand the critical safety mechanism - the alarm MUST automatically stop at the specified end time regardless of puzzle completion status.

Implement the automatic alarm termination system:
- Precise end-time monitoring and enforcement
- Automatic service shutdown at end time
- Cleanup routines for audio resources
- User notification of auto-stop occurrence
- Statistics tracking for auto-stopped vs solved alarms
- Grace period handling (30-second buffer for processing)
- End-time validation and error handling
- Recovery scenarios if end-time check fails

Prompt 12: System Integration & Permissions

Handle Android system integration requirements:
- Runtime permissions (WAKE_LOCK, FOREGROUND_SERVICE, etc.)
- Battery optimization whitelist guidance for users
- Do Not Disturb mode integration and override

- Lock screen integration and display
- System alert window permissions for overlay
- Boot completion receiver for alarm restoration
- Notification channel setup and management
- Handle permission denial scenarios gracefully

Phase 7: Testing & Build Preparation

Prompt 13: Create Debug Build & Basic Testing

Prepare the app for initial testing:
- Create debug build configuration
- Set up logging and debugging tools
- Create test scenarios for core functionality
- Add developer options for testing (quick alarm, skip puzzle mode)
- Create APK for device testing
- Include crash reporting setup (Firebase Crashlytics)
- Add performance monitoring
- Create testing checklist for core features

Prompt 14: Device Testing & Feedback Integration

Prepare for multi-device testing phase:
- Create feedback collection mechanism within app
- Add device information reporting for bug reports
- Create different build variants (debug, staging, release)
- Set up remote logging for beta testing
- Add in-app testing tools (alarm simulation, service status)
- Create user guide for beta testers
- Implement easy bug reporting feature
- Add performance metrics collection

Phase 8: Refinement & Optimization

Prompt 15: Performance Optimization

Optimize app performance and reliability:
- Memory usage optimization
- Battery consumption analysis and reduction
- Audio latency minimization
- Service restart speed optimization
- Database query optimization
- UI responsiveness improvements

- Background processing efficiency
- Resource cleanup and leak prevention

Prompt 16: Edge Case Handling

Refer to alarmise_context.md to ensure all edge case solutions maintain the core principles: non-stop alarm, math-only dismissal, and automatic end-time termination.

Handle complex edge cases and scenarios:
- Multiple alarms conflict resolution
- System time changes (DST, manual changes)
- Low storage space scenarios
- Network connectivity issues (if any features require it)
- App update scenarios while alarm is active
- Factory reset recovery
- Corrupted database recovery
- Audio hardware failure fallbacks

Phase 9: Production Readiness

Prompt 17: Security & Privacy Implementation

Implement security and privacy features:
- Data encryption for alarm storage
- Privacy policy compliance
- No unnecessary permissions requests
- Secure audio file handling
- User data protection measures
- Compliance with Google Play policies
- GDPR compliance if applicable
- Security audit checklist

Prompt 18: Final Polish & Release Preparation

Before finalizing, thoroughly review alarmise_context.md to validate that all success criteria and non-negotiable rules are properly implemented.

Prepare for production release:
- Final UI polish and consistency check
- Complete error message review
- User onboarding flow creation
- App store listing preparation (screenshots, description)
- Create release build configuration
- Final testing on multiple devices and Android versions

```
- Performance benchmark establishment
- Create user documentation and FAQ
```

Phase 10: Production Build & Deployment

Prompt 19: Production Build Creation

```
Create the production-ready build:
- Release build configuration with proper signing
- ProGuard/R8 optimization setup
- Play Store release preparation
- Version management and changelog
- Final security review
- Performance validation
- Create signed APK/AAB for distribution
- Beta testing group setup if needed
```

Prompt 20: Post-Launch Monitoring Setup

```
Set up post-launch monitoring and maintenance:
- Crash reporting analysis setup
- User feedback monitoring system
- Performance metrics dashboard
- Update mechanism planning
- Bug fix prioritization system
- User support workflow
- App store rating monitoring
- Future feature planning framework
```

Development Tips for Each Phase:

General Guidelines:

- Test each prompt's implementation thoroughly before moving to the next
- Always ask Claude to explain any complex technical decisions
- Request code comments and documentation for critical components
- Ask for alternative approaches when facing technical challenges
- Validate that each feature aligns with the original requirements

Key Testing Points:

- **After Prompt 7**: Test basic audio playback

- **After Prompt 11**: Test complete alarm cycle
- **After Prompt 13**: Create first APK for device testing
- **After Prompt 16**: Comprehensive testing phase
- **After Prompt 19**: Final production build

Critical Success Checkpoints:

1. **Prompt 6 completion**: Service runs in background
2. **Prompt 8 completion**: Audio survives app termination
3. **Prompt 10 completion**: Math puzzle dismissal works
4. **Prompt 11 completion**: Auto-stop mechanism functions
5. **Prompt 13 completion**: Full cycle testing successful

Remember: Each prompt should be followed by testing and validation before proceeding to ensure the core reliability requirements are met at every step.