AskMe Lite Tech Stack Evolution Summary

📊 Compatibility Matrix

Component	Initial Suggestion	Manager Feedback	Final Recommendation	Final Version
JDK	17.0.10+ (LTS)	✅ Approved	Maintain	17
Kotlin	1.9.22	❌ **CRITICAL** - Compatibility issues with Compose	Downgrade for stability	**1.9.10**
Gradle	8.2+	⚠️ **HIGH** - Suboptimal compatibility	Upgrade for better alignment	**8.4**
Android Gradle Plugin	8.2.2+	✅ Approved with adjustment	Stable version	**8.1.4**
Jetpack Compose	1.6.7+ (individual versions)	❌ **CRITICAL** - Version conflicts	Implement BOM approach	**BOM 2023.10.01**
Compose Compiler	Not specified	Required for alignment	Match Kotlin version	**1.5.4**
Android SDK	API 34 target, min 24	✅ Approved	Maintain	API 34/24
Android NDK	25.1.8937393	✅ Approved	Maintain	25.1.8937393
AndroidX Security	1.1.0-alpha06	⚠️ **MEDIUM** - Stability risk	Use more stable alpha	**1.1.0-alpha04**
Ktor	2.3.7	✅ Approved with adjustment	Stable version	**2.3.6**
gRPC Kotlin	1.60+	✅ Approved with adjustment	Stable version	**1.4.1**
gRPC Java	1.60+	✅ Approved with adjustment	Stable runtime	**1.58.0**
Koin	3.5.6	✅ Approved with adjustment	Proven stable	**3.5.0**
SQLDelight	2.0.1	✅ Approved with adjustment	Stable 2.x branch	**2.0.0**
Detekt	1.23.6	✅ Approved with adjustment	Stable analysis	**1.23.4**
JUnit 5	5.10.2	✅ Approved with adjustment	Stable version	**5.10.1**
MockK	1.13.10	✅ Approved with adjustment	Stable mocking	**1.13.8**
Kotest	5.8.0	✅ Approved	Maintain	**5.8.0**
Dokka	1.9.20	Adjusted to match Kotlin	Match Kotlin version	**1.9.10**

📝 Decision Journey Summary

Phase 1: Initial Assessment & Gaps Identified

The initial audit revealed a critical gap: **most component versions were unspecified** in the project documentation, creating a high risk of "version hell." The team had experienced build failures due to version mismatches, necessitating a comprehensive compatibility review.

Phase 2: Manager's Critical Feedback

The manager identified **three severity levels of issues**:

- **CRITICAL**: Kotlin 1.9.22 + Compose 1.6.7 incompatibility causing build failures and runtime crashes
- **HIGH**: Gradle 8.2.1 + Kotlin 1.9.22 mismatch leading to slower builds and plugin resolution issues
- **MEDIUM**: Production use of AndroidX Security alpha06 creating stability risks

Phase 3: Strategic Resolution

Key architectural decisions made:

- **Implemented Compose BOM strategy** instead of individual library versioning to prevent version conflicts
- **Prioritized stability over cutting-edge versions** for production deployment
- **Established strict version alignment** between Kotlin, Compose, and Gradle ecosystems
- **Reduced alpha dependency risks** while maintaining necessary functionality

Phase 4: Production-Ready Implementation

The final implementation includes:

- **Comprehensive version catalog** with detailed rationale for each decision
- **Bundle definitions** for logical grouping of related dependencies
- **Plugin version alignment** ensuring consistent toolchain behavior
- **Documentation of upgrade paths** and compatibility references

🎯 Key Trade-offs & Rationale

Stability vs. Innovation

- **Trade-off**: Chose slightly older but proven stable versions over bleeding-edge releases
- **Rationale**: Production reliability prioritized over latest features, especially after experiencing version-related build failures

Kotlin Version Downgrade (1.9.22 → 1.9.10)

- **Impact**: Sacrificed some newer language features for ecosystem stability

- **Benefit**: Eliminated critical compatibility issues with Compose and multiplatform tooling
- **Reference**: Compose-Kotlin Compatibility Table

Compose BOM Implementation

- **Change**: Shifted from individual library versioning to Bill of Materials approach
- **Benefit**: Eliminates version conflicts across Compose libraries automatically
- **Impact**: Reduces maintenance overhead and prevents subtle runtime errors

Alpha Dependency Management

- **Constraint**: AndroidX Security only available in alpha releases
- **Solution**: Used most stable alpha (alpha04) instead of latest (alpha06)
- **Monitoring**: Requires ongoing assessment for stable release availability

🔄 Maintenance Strategy

The final tech stack includes:

- **Automated compatibility checking** through version catalogs
- **Clear upgrade documentation** with official reference links
- **Regular review schedule** tied to release cycles
- **Rollback procedures** for problematic version combinations

This evolution demonstrates a mature approach to dependency management, balancing innovation with operational stability while establishing processes to prevent future version conflicts.