📋 **Zusammenfassung der Analyse**

Ich habe die PDF-Aufgabenstellung analysiert und ein detailliertes Dokument erstellt, das folgende Bereiche abdeckt:

1. Code-Analyse der iOS App
- **Architektur-Pattern**: Modulare Struktur mit Domain, Infrastructure und Utility Layers
- **Wichtige Features**: Async/Await, Network Service, Storage Service, Hybrid Data Loading
- **Datenfluss**: Sync-Prozess, Caching, Update-Mechanismus, Error Handling

2. Empfohlene Vorgehensweise
- **Phase 1**: Domain Driven Design Setup mit Kotlin
- **Phase 2**: Infrastructure Layer (Ktor + Room Database)
- **Phase 3**: Presentation Layer (Jetpack Compose)
- **Phase 4**: Dependency Injection (Hilt)

3. Technologie-Stack
- **Core Libraries**: Jetpack Compose, Kotlin Coroutines, Hilt, Room, Ktor
- **Architecture Components**: ViewModel, Repository Pattern, Use Cases, StateFlow

4. Migration-Strategie
- Bottom-Up Approach
- Feature-by-Feature Migration
- Comprehensive Testing Strategy

📁 **Erstellte Dateien**

1. **`Analyse_iOS_zu_Android_Migration.md`** - Markdown-Version
2. **`Analyse_iOS_zu_Android_Migration.html`** - HTML-Version (kann in Word konvertiert werden)

🔄 **Nächste Schritte**

Das HTML-Dokument können Sie einfach in Microsoft Word öffnen und als .docx speichern. Die Analyse bietet Ihnen:

- **Detaillierte Code-Beispiele** für die Android-Implementierung
- **Schritt-für-Schritt Anleitung** für die Migration
- **Architektur-Empfehlungen** basierend auf Domain Driven Design
- **Technologie-Vergleiche** zwischen iOS und Android

Die Analyse berücksichtigt die spezifischen Anforderungen Ihrer iOS App und bietet eine solide Grundlage für die Android-Entwicklung mit Jetpack Compose.