Realisierung

Delegator

Klassifizierung	intern
Status	in Arbeit
Programmname	Delegator
Projektnummer	1
Projektleiter	Tabinas Kenan
Version	0.1
Datum	26. März 2025
Auftraggeber	Tabinas Kenan
Autor/Autoren	Tabinas Kenan
Verteiler	

Änderungsverzeichnis

Version	Datum	Änderung	Autor
0.1			

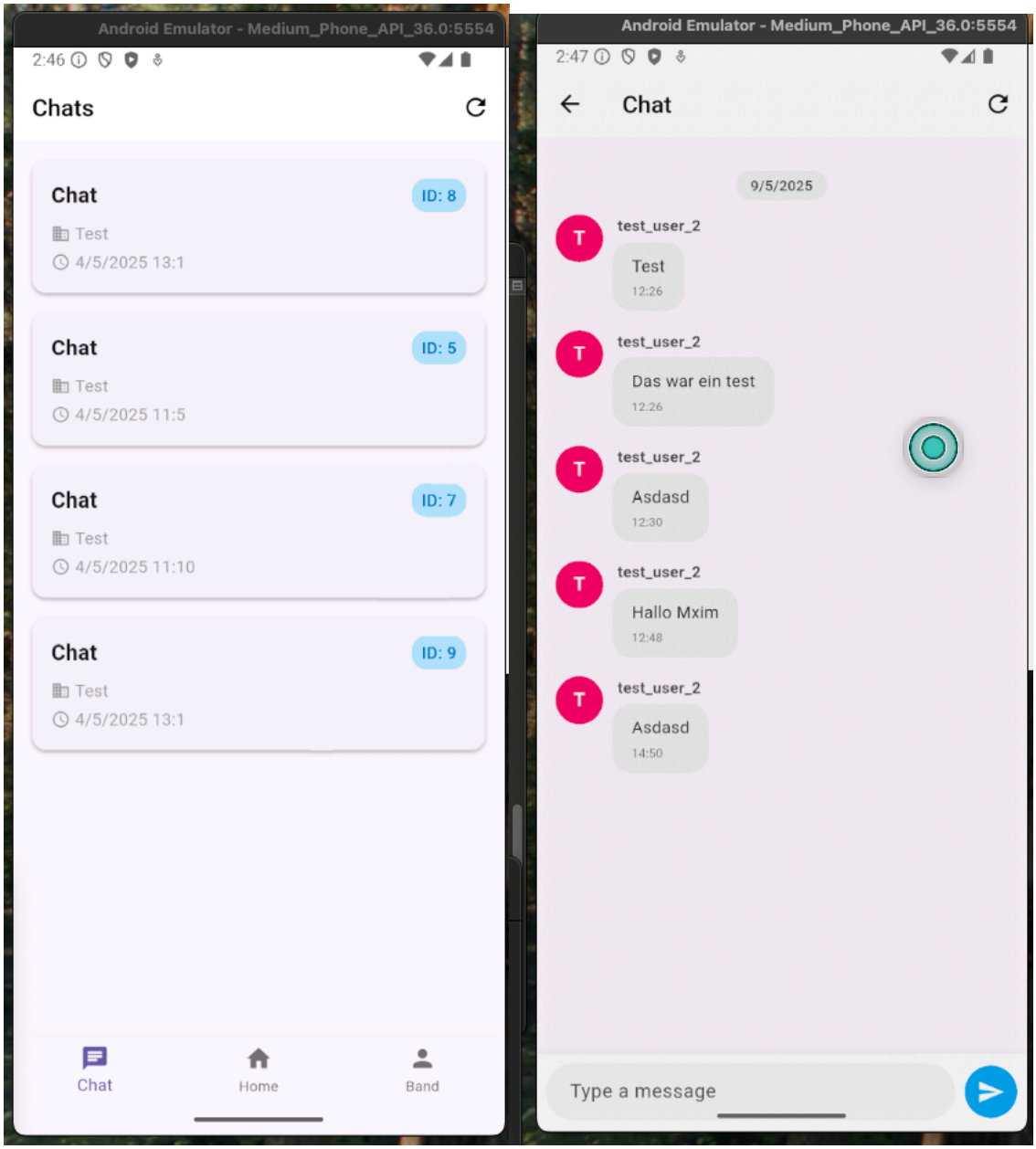
Tabelle 1: Änderungsverzeichnis

Beschreibung

Die Studie beschreibt die angestrebte Lösung, in dem sie die groben Ziele definiert, möglichen Lösungsvarianten aufführt und diese dann bewertet. Sie bildet die Grundlage für die Entscheidung, ob ein Projekt freigegeben wird oder nicht. Sie ist die Voraussetzung für die Erarbeitung des Projektmanagementplans und des Projektauftrags.

1 Designpattern in Flutter

Zuerst habe einfach angefangen, weil ich wenig Erfahrung mit Flutter hatte. Ich habe vieles ChatGPT / Claude machen lassen. Das zu einem gewissen Grad funktioniert. Der Chat Teil hat wunderbar funktioniert.



1.1 Design Pattern

Im Verlauf hat sich das leider nicht tragbar gezeigt. Beim erstellen von Tasks bin ich an die grenzen gekommen. Die Language Modells konnten mir nicht mehr helfen. Selbst mit Zurgiff auf das Repository, keine Chance. Der Code scheint gut auszusehen, sagten sie. Irgendwelche Lösung Ansätze die nicht weiter brachten.

Ich habe Claude nach Design Patterns gefragt folgende Vorschläge gab es:

Aspekt	Feature-Based	Clean Architecture	MVVM
Grundprinzip	Organisation nach Funktionalitäten	Strenge Schichtentrennung mit Abhängigkeitsregeln	Trennung von Ansicht und Ansichtslogik
Ordnerstruktur	Nach Features gruppiert, jedes Feature enthält alle relevanten Schichten	Konzentrische Kreise: Domain (innerste), Data, Presentation (äußerste)	Models, Views, ViewModels als Hauptgruppen
Stärken	Leichte Navigation innerhalb eines Features, bessere Isolation von Features, gut für parallele Teamarbeit	Sehr klare Abhängigkeitsrichtung, Domain-Schicht völlig unabhängig und Langfristige Wartbarkeit	Intuitive Struktur, einfacher zu erlernen und Gute Testbarkeit der ViewModels
Schwächen	Mögliche Codeduplizierung zwischen Features, kann bei gemeinsamen Komponenten unübersichtlich werden	Steilere Lernkurve, Mehr Boilerplate-Code und Überkompliziert für kleinere Apps	Weniger rigide Regeln als Clean Architecture, ViewModels können zu groß werden
Skalierbarkeit	Sehr gut	Exzellent	Gut
Komplexität	Mittel	Hoch	Mittel
Eignung für	Mittelgroße bis große Apps mit klar abgegrenzten Features	Komplexe, langlebige Enterprise-Anwendungen	Apps mit vielen UI-Zuständen
State Management	Flexibel (BLoC, Provider, Riverpod, usw.)	Meist Use-Cases und Repositories	Typischerweise Provider oder Riverpod
Testbarkeit	Gut	Sehr gut	Gut

Code-Isolation	Feature-Isolation	Schichten-Isolation	Zuständigkeits-Isolation
Lernkurve	Mittel	Steil	Flach bis mittel
Teamarbeit	Sehr gut (Feature-Teams)	Gut (Schichten-Teams)	Gut
Änderungen an Geschäftslogik	Auf Features beschränkt	Einfach durch isolierte Domain-Schicht	Auf ViewModels beschränkt
Änderungen an UI	Auf Features beschränkt	Auf Presentation-Schicht beschränkt	Nur Views betroffen
Wiederverwendbarkeit	Mäßig zwischen Features	Sehr gut (besonders Domain-Schicht)	Gut (besonders ViewModels)
Code-Organisation	Nach Business-Funktionen	Nach technischen Schichten	Nach Presentation-Pattern
Flutter-Spezifisch	Gut integrierbar	Erfordert mehr Anpassung	Natürlich passend

1.2 Nutzwertanalyse

Habe anschliessend eine Wertung gemacht. Basierend auf mein Projekt und folgender Wert:

Wert	Bedeutung
0	Irrelvant
1	Schlecht
2	Mittel
3	Gut

Tabelle 2: Wertung

Aspekt	Feature-Based	Clean Architecture	MVVM
Grundprinzip	0	0	0
Ordnerstruktur	0	0	0
Stärken	1	2	3
Schwächen	2	1	2
Skalierbarkeit	3	3	2
Komplexität	2	1	2
Eignung für	3	1	2
State Management	0	0	0
Testbarkeit	2	3	2
Code-Isolation	0	0	0
Lernkurve	2	1	2
Teamarbeit	0	0	0
Änderungen an Geschäftslogik	0	0	0
Änderungen an UI	0	0	0
Wiederverwendbarkeit	1	3	3
Code-Organisation	0	0	0
Flutter-Spezifisch	2	1	3
TOTAL	**18**	**16**	**21**

Tabelle 3: Pattern Nutzwertanalyse

1.3 Fazit

Nach der Wertung hat klar MVVM gewonnen und so fahre ich fort. Beim ersten Anlauf ist mir Aufgefallen, das der Service, also die Schnittstelle zum Backend am Problematischstens ist. Daher habe ich mich folgenden Plan erstellt.

- Services
- Models
- Views
- Integrationstest

Inhaltsverzeichnis

Abbildungsverzeichnis

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

Tabellenverzeichnis