

TaskMaster

Eine Aufgabenverwaltungs-App mit Flutter, Firebase und BLoC

Nicolas Ewald

Übersicht

1. Projektübersicht
 - Warum TaskMaster?
 - Ziel der App
2. Prototyp von TaskMaster
3. Technologie-Stack
 - Dart
 - Flutter
 - Firebase
 - BLoC
 - BLoC Beispiel
4. Architektur
5. Die UI
6. Firestore Datenbankstruktur
7. Ausblick
8. Fragen

Warum TaskMaster?

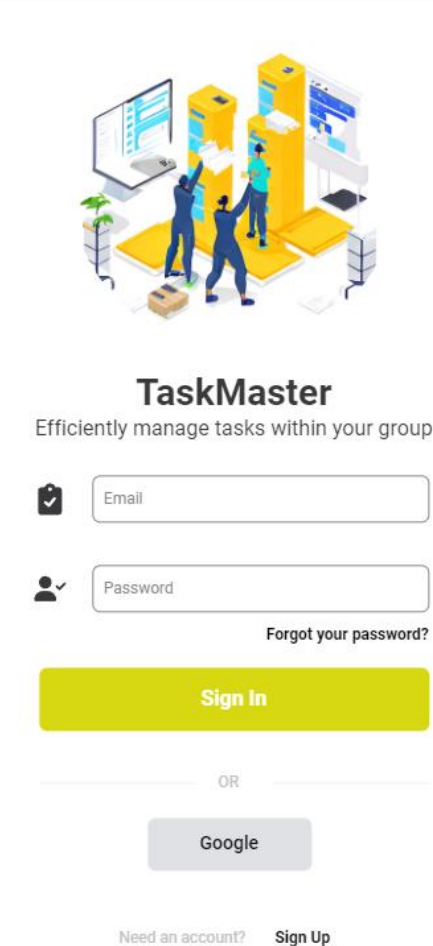
- Herausforderung im Betrieb:
 - Ineffiziente Aufgabenzuweisung und -verwaltung im Betrieb
 - Aufgaben wurden oft nur mündlich oder per Post-its weitergegeben
 - Mangel an strukturierter Verfolgung und Priorisierung
- Inspiration:
 - Diese Herausforderungen brachten mich auf die Idee, eine App zur Aufgabenverwaltung zu entwickeln

Ziel der App

- Effiziente Aufgabenverwaltung:
 - Unterstützung für kleine und große Betriebe sowie Privatpersonen
 - Optimierung der Zuweisung, Verfolgung und Priorisierung von Aufgaben
- Produktivitätssteigerung:
 - Zentralisierte Plattform für alle Aufgaben
 - Einfache und klare Übersicht über alle anstehenden Aufgaben
 - Priorisierung von Aufgaben nach Wichtigkeit und Dringlichkeit

Prototyp - Authentifizierung

- Login-Screen:
 - Benutzeranmeldung
 - Wechsel zu Profil erstellen
- Profil erstellen:
 - Einfaches Erstellen eines Benutzerprofils
 - Benutzername und Passwort festlegen



The login screen for TaskMaster features a header illustration of people working at computers. Below this, the app name 'TaskMaster' is displayed with the tagline 'Efficiently manage tasks within your group'. The login form includes an 'Email' field with a checkmark icon, a 'Password' field with a checkmark icon, and a 'Forgot your password?' link. A prominent yellow 'Sign In' button is centered below the fields. An 'OR' separator is followed by a grey 'Google' button. At the bottom, a link for 'Need an account? Sign Up' is provided.

TaskMaster
Efficiently manage tasks within your group

Email

Password

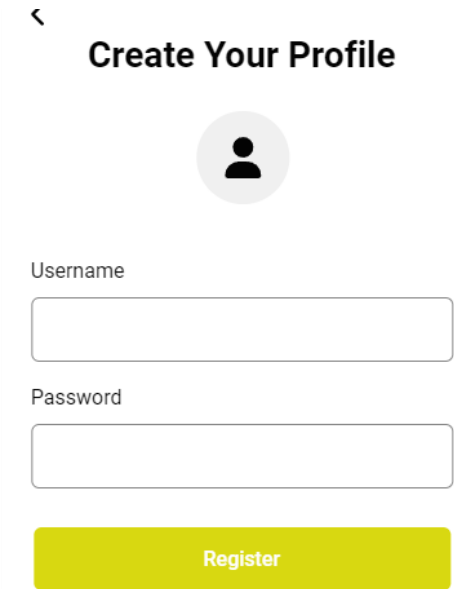
[Forgot your password?](#)

Sign In

OR


Google

[Need an account? Sign Up](#)



The 'Create Your Profile' screen has a back arrow in the top left. It features a circular profile icon placeholder. Below this are input fields for 'Username' and 'Password'. A large yellow 'Register' button is positioned at the bottom of the form.

< **Create Your Profile**



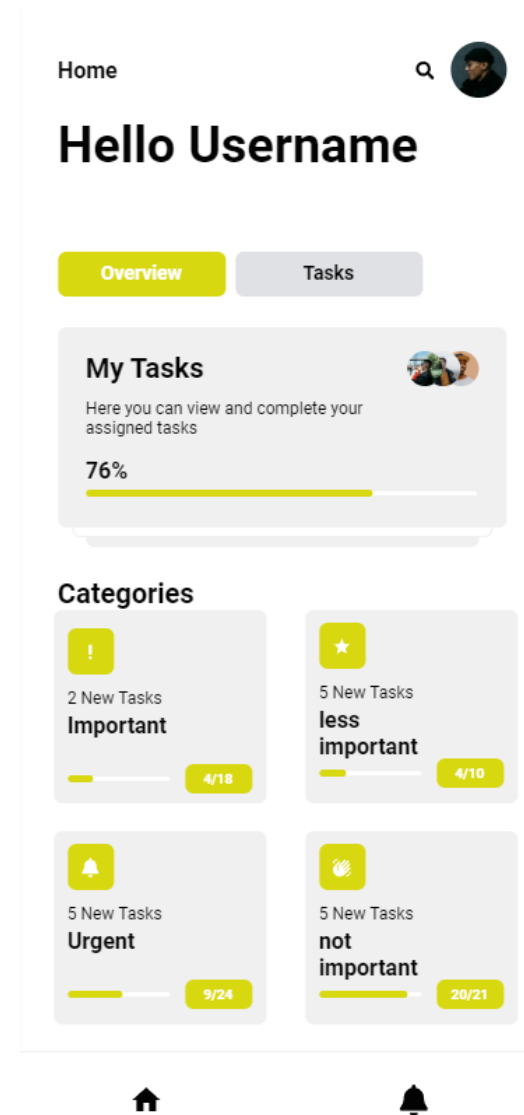
Username

Password

Register

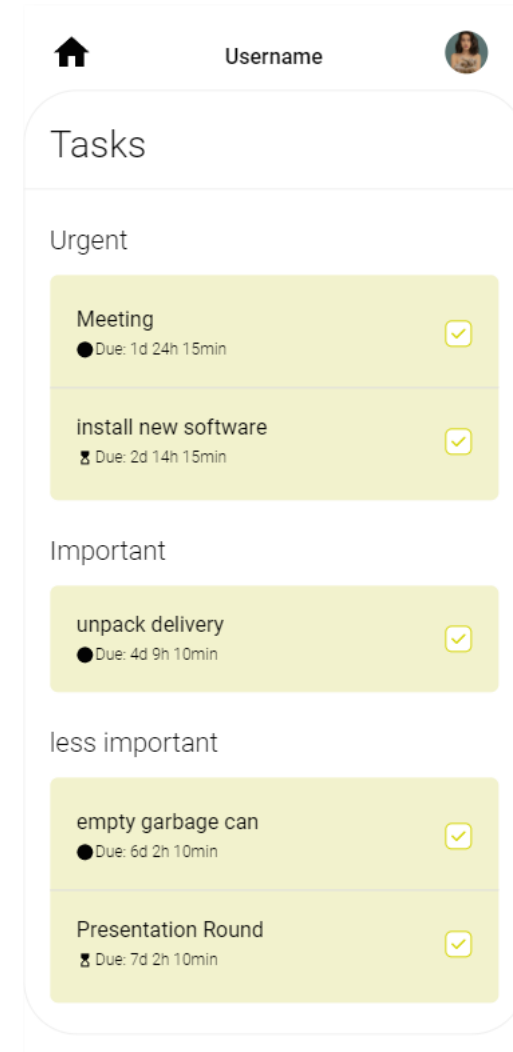
Prototyp – der Homescreen

- Übersicht über Aufgaben:
 - Zeigt den Fortschritt und den Status der zugewiesenen Aufgaben.
- Kategorisierung:
 - Aufgaben nach Wichtigkeit kategorisiert.
- Benutzerfreundliche Navigation



Prototyp – Tasks

- Kategorisierung von Aufgaben:
 - Aufgaben werden nach Dringlichkeit kategorisiert.
- Detaillierte Aufgabenübersicht:
 - Zeigt Aufgabenname und Zeit bis zur Fälligkeit an.
 - Visuelle Markierungen für erledigte Aufgaben.



Prototyp - Gruppen

- Gruppenübersicht:
 - Anzeige von Gruppen mit Namen und Mitgliederanzahl.
 - Beschreibung der jeweiligen Gruppe.
- Aufgaben zuweisen:
 - Einfaches Zuweisen von Aufgaben an Gruppenmitglieder.
 - Eingabefelder für Titel, Beschreibung, Fälligkeitsdatum und Priorität.
 - Auswahl der Gruppenmitglieder, die die Aufgabe übernehmen sollen.

←

Group List

Design Team 4 Members ✓
A group for creative minds

Marketing Campaign 8 Members ✓
Strategies for our next campaign

Product Development 6 Members ✓
Innovating new products

Home Notification

← AdminUser +

Assign Tasks

Task Title

Enter task title

Task Description

Enter task description

Due Date

dd/mm/yy

Priority Level

Low

Assign Task

Select Group Members

☐ Group Member 1

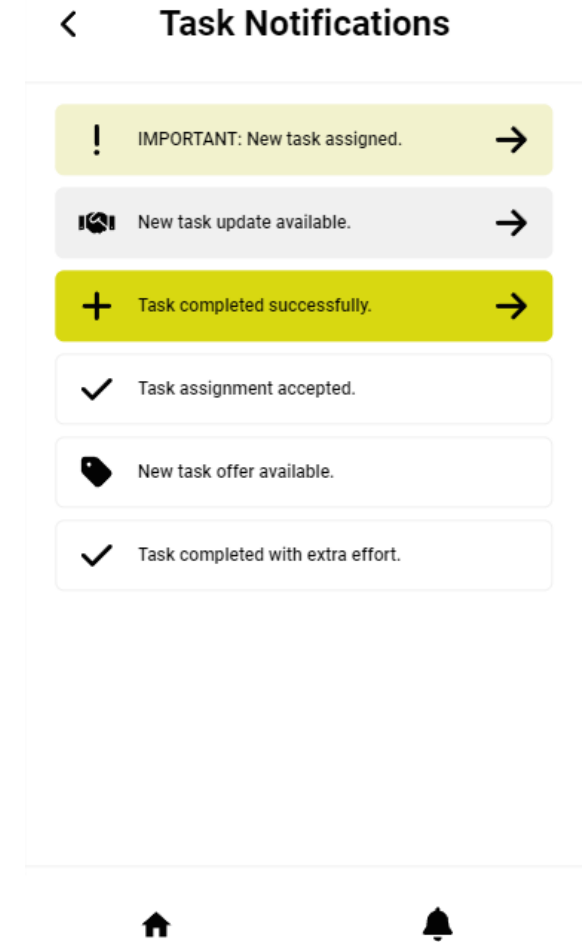
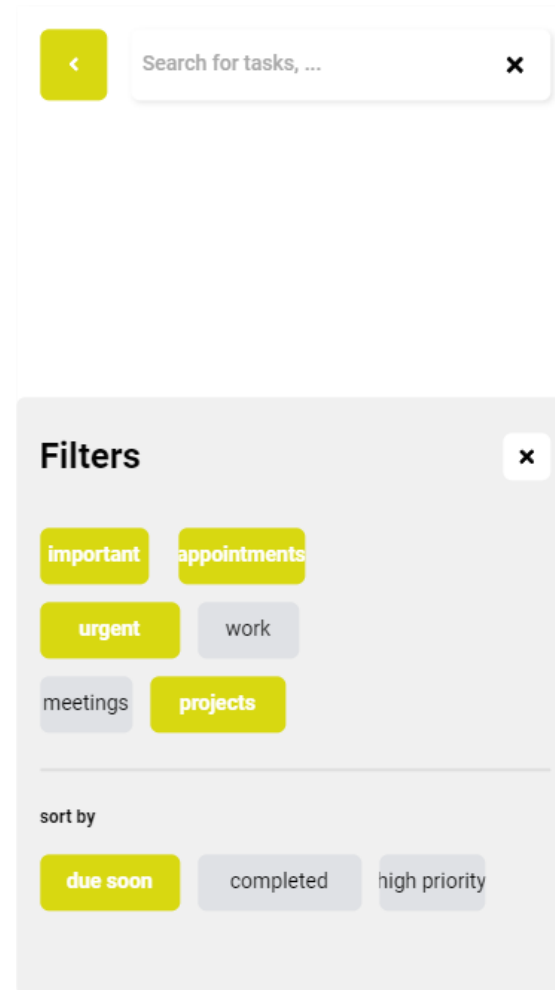
☐ Group Member 2

☐ Group Member 3

Home Notification

Prototyp – weitere Funktionen

- Such- und Filterfunktionen:
 - Benutzer können Aufgaben anhand von Stichwörtern durchsuchen.
 - Filteroptionen, um Aufgaben nach Wichtigkeit, Dringlichkeit oder Kategorie zu sortieren.
- Benachrichtigungen:
 - Echtzeit-Benachrichtigungen über neue Aufgaben und Updates.



Dart

- objektorientierte Programmiersprache
- von Google entwickelt
- speziell für die Entwicklung von mobilen und Web-Anwendungen
- Einfache Syntax
- Unterstützt sowohl AOT (Ahead-of-Time) als auch JIT (Just-in-Time) Kompilierung.

Flutter

- ist ein Open-Source UI-Toolkit von Google

Vorteile:

- Einheitliche Codebasis: Einmal schreiben, überall ausführen – sowohl für iOS als auch für Android, Web und Desktop.
- Flexibilität und Anpassbarkeit: Mit einer breiten Palette an Widgets und umfangreichen Anpassungsmöglichkeiten.
- Schnelle Entwicklung: Durch die "Hot Reload" Funktion können Entwickler Änderungen sofort sehen, was die Entwicklungszeit erheblich reduziert.

Firebase

- Ist eine Plattform von Google
- Firebase bietet Backend-Dienste wie
 - Firebase Auth:
 - Bietet einfache Authentifizierungsmethoden wie E-Mail/Passwort, Google-Anmeldung, Facebook-Anmeldung und mehr.
 - Cloud Firestore:
 - Echtzeit-Datenbank, die synchronisierte Datenverwaltung über alle Clients hinweg ermöglicht.
 - Und vieles mehr

BLoC (Business Logic Component)

- BLoC ist ein Architektur-Pattern für Flutter, das die Trennung von Geschäftslogik und UI ermöglicht.
- Kernkonzepte von BLoC:
 1. Event:
 - Repräsentiert Benutzeraktionen oder Ereignisse, die eine Zustandsänderung auslösen.
 - Beispiele: Button-Klick, Datenanforderung.
 2. State:
 - Repräsentiert den aktuellen Zustand der UI.
 - Beispiele: Ladezustand, Erfolgszustand, Fehlerzustand.
 3. Bloc:
 - Enthält die Business-Logik.
 - Nimmt Events entgegen, verarbeitet diese und gibt neue Zustände zurück.
 - Besteht aus einem Stream von Events und einem Stream von Zuständen.

BLoC – Beispiel CounterApp

- Ursprüngliche Implementierung:

- StatefulWidget: Die Logik zum Zählen befindet sich direkt in der UI-Komponente.
- setState: Wird verwendet, um den Zustand zu aktualisieren und die UI neu zu rendern.
- Enge Kopplung: Die Logik und die UI sind stark gekoppelt, was die Wiederverwendbarkeit und Testbarkeit erschwert.

- Bloc-Implementierung:

- StatelessWidget: Die UI-Komponenten sind stateless und erhalten den Zustand über den Bloc.
- BlocProvider: Wird verwendet, um den Bloc im Widget-Baum bereitzustellen.
- BlocBuilder: Verwendet den Zustand des Blocs, um die UI zu rendern.
- Entkopplung: Die Logik ist von der UI getrennt, was die Wiederverwendbarkeit und Testbarkeit verbessert.

```
9  @override
10 Widget build(BuildContext context) {
11   return MaterialApp(
12     title: 'Flutter Demo',
13     theme: ThemeData(
14       colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
15       useMaterial3: true,
16     ), // ThemeData
17   home: const MyHomePage(title: 'Flutter Demo Home Page'),
18 ); // MaterialApp
19 }
20 }

41 @override
42 Widget build(BuildContext context) {
43   return Scaffold(
44     appBar: AppBar(
45       backgroundColor: Theme.of(context).colorScheme.inversePrimary,
46       title: Text(widget.title),
47     ), // AppBar
48     body: Center(
49       child: Column(
50         mainAxisAlignment: MainAxisAlignment.center,
51         children: <Widget>[
52           const Text(
53             'You have pushed the button this many times',
54           ), // Text
55           Text(
56             '$_counter',
57             style: Theme.of(context).textTheme.headlineMedium,
58           ), // Text
59         ], // <Widget>[]
60       ), // Column
61     ), // Center
62     floatingActionButton: FloatingActionButton(
63       onPressed: _incrementCounter,
64       tooltip: 'Increment',
65       child: const Icon(Icons.add),
66     ), // FloatingActionButton
67   ); // Scaffold
68 }
69 }
```

```
12 @override
13 Widget build(BuildContext context) {
14   return MaterialApp(
15     title: 'Flutter Demo',
16     theme: ThemeData(
17       colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
18       useMaterial3: true,
19     ), // ThemeData
20+   home: BlocProvider(
21+     create: () => CounterBloc(),
22+     child: const MyHomePage(title: 'Flutter Demo Home Page'),
23+   ), // BlocProvider
24 ); // MaterialApp
25 }
26 }

33 @override
34 Widget build(BuildContext context) {
35   return Scaffold(
36     appBar: AppBar(
37       backgroundColor: Theme.of(context).colorScheme.inversePrimary,
38+     title: Text(title),
39   ), // AppBar
40   body: Center(
41     child: Column(
42       mainAxisAlignment: MainAxisAlignment.center,
43       children: <Widget>[
44         const Text(
45           'You have pushed the button this many times:',
46         ), // Text
47         BlocBuilder<CounterBloc, CounterState>{
48           builder: (context, state) {
49             int count = state is CounterUpdated ? state.count : 0;
50             return Text(
51               '$count',
52               style: Theme.of(context).textTheme.headlineMedium,
53             ); // Text
54           }, // BlocBuilder
55         ], // <Widget>[]
56       ), // Column
57     ), // Center
58     floatingActionButton: FloatingActionButton(
59       onPressed: () => context.read<CounterBloc>().add(Increment()),
60       tooltip: 'Increment',
61       child: const Icon(Icons.add),
62     ), // FloatingActionButton
63   ); // Scaffold
64 }
```

BLoC - Implementierung

1. Event:

- CounterEvent:
Eine abstrakte Basisklasse für alle Events, die der CounterBloc verarbeiten kann.
- Increment:
Ein konkretes Event, das von der Benutzeraktion „Increment“ repräsentiert wird.

2. State:

- CounterState:
Eine abstrakte Basisklasse für alle Zustände, die der CounterBloc haben kann.
- CounterInitial:
Ein Zustand, der den initialen Zustand des Zählers darstellt. Er enthält die Variable „count“, die den aktuellen Zählerstand speichert.
- CounterUpdated:
Ein Zustand, der den aktualisierten Zählerstand nach einem Increment-Event speichert.

3. Bloc

- Constructor:
Initialisiert den Bloc mit dem CounterInitial-Zustand und setzt den Anfangszähler auf 0.
- Logik:
Prüft den aktuellen Zustand: Wenn der Zustand CounterInitial ist, wird der count auf 0 gesetzt.
Andernfalls (wenn der Zustand CounterUpdated ist), wird der count aus dem aktuellen Zustand übernommen.
Der count wird dann um 1 erhöht.
emit: Emittiert den neuen Zustand CounterUpdated mit dem aktualisierten count.

```
1  import 'package:bloc/bloc.dart';
2
3  // Event-Klassen
4  abstract class CounterEvent {}
5
6  class Increment extends CounterEvent {}
7
8  // State-Klassen
9  abstract class CounterState {}
10
11 class CounterInitial extends CounterState {
12   final int count;
13   CounterInitial(this.count);
14 }
15
16 class CounterUpdated extends CounterState {
17   final int count;
18   CounterUpdated(this.count);
19 }
20
21 // Bloc-Klasse
22 class CounterBloc extends Bloc<CounterEvent, CounterState> {
23   CounterBloc() : super(CounterInitial(0)) {
24     on<Increment>(_onIncrement);
25   }
26
27   void _onIncrement(Increment event, Emitter<CounterState> emit) {
28     final newCount =
29       (state is CounterInitial ? 0 : (state as CounterUpdated).count) + 1;
30     emit(CounterUpdated(newCount));
31   }
32 }
```

Architektur

„Clean Architecture“ ist ein Architekturstil für den Aufbau einer Anwendung.

Strukturierter, testbarer und nachvollziehbarer.

blocs:

- enthält die Business Logic Components (BLoCs).

core:

- grundlegende, wiederverwendbare Komponenten und Dienste der Anwendung.

domain:

- repräsentiert die Kernlogik der Anwendung.

infrastructure:

- Implementierungen der Repository-Interfaces und andere Datenzugriffsschichten. APIs, Datenbankzugriffe und externe Dienste.

presentation:

- enthält alle UI-bezogenen Komponenten wie Widgets und Screens

firebase_options.dart:

- enthält die Firebase-Konfigurationsoptionen. Wird durch das Firebase CLI-Tool generiert und enthält Einstellungen, die für die Verbindung mit Firebase-Diensten erforderlich sind.

injection.dart:

- ist für die Dependency Injection zuständig. Hier werden Abhängigkeiten registriert und bereitgestellt, damit sie in der gesamten Anwendung verwendet werden können.

main.dart:

- Der Einstiegspunkt der Anwendung. Enthält die main-Funktion und die Grundkonfiguration der App. Hier wird die Anwendung initialisiert und gestartet.

```
✓ lib
  > blocs
  > core
  > domain
  > infrastructure
  > presentation
  🔍 firebase_options.dart
  🔍 injection.dart
  🔍 main.dart
```

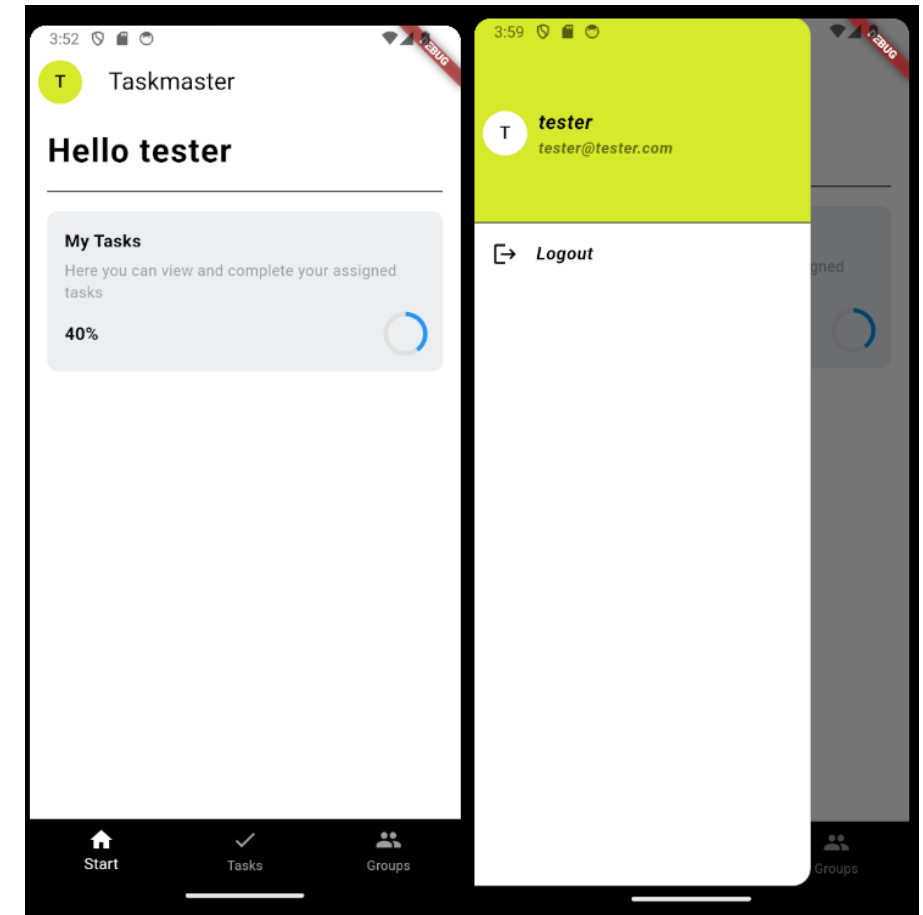
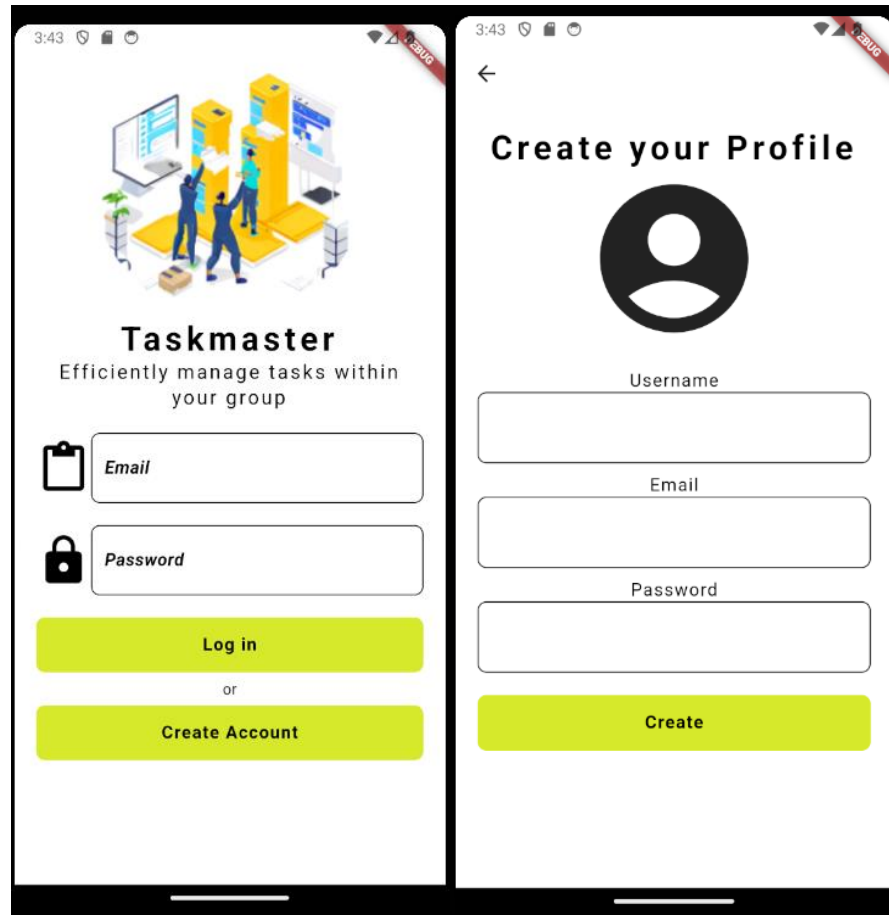
```
✓ lib
  ✓ blocs
    > auth
    > group
    > user_task
  ✓ core
    > buttons
    > errors
    > extensions
    > failures
    > utils
  ✓ domain
    > entities
    > repositories
  ✓ infrastructure
    > extensions
    > models
    > repositories_impl
  ✓ presentation
    > routes
    > screens
  🔍 firebase_options.dart
  🔍 injection.dart
  🔍 main.dart
```


Die UI – Authentifizierung & Home

Login

Registration

Home Screen & Drawer



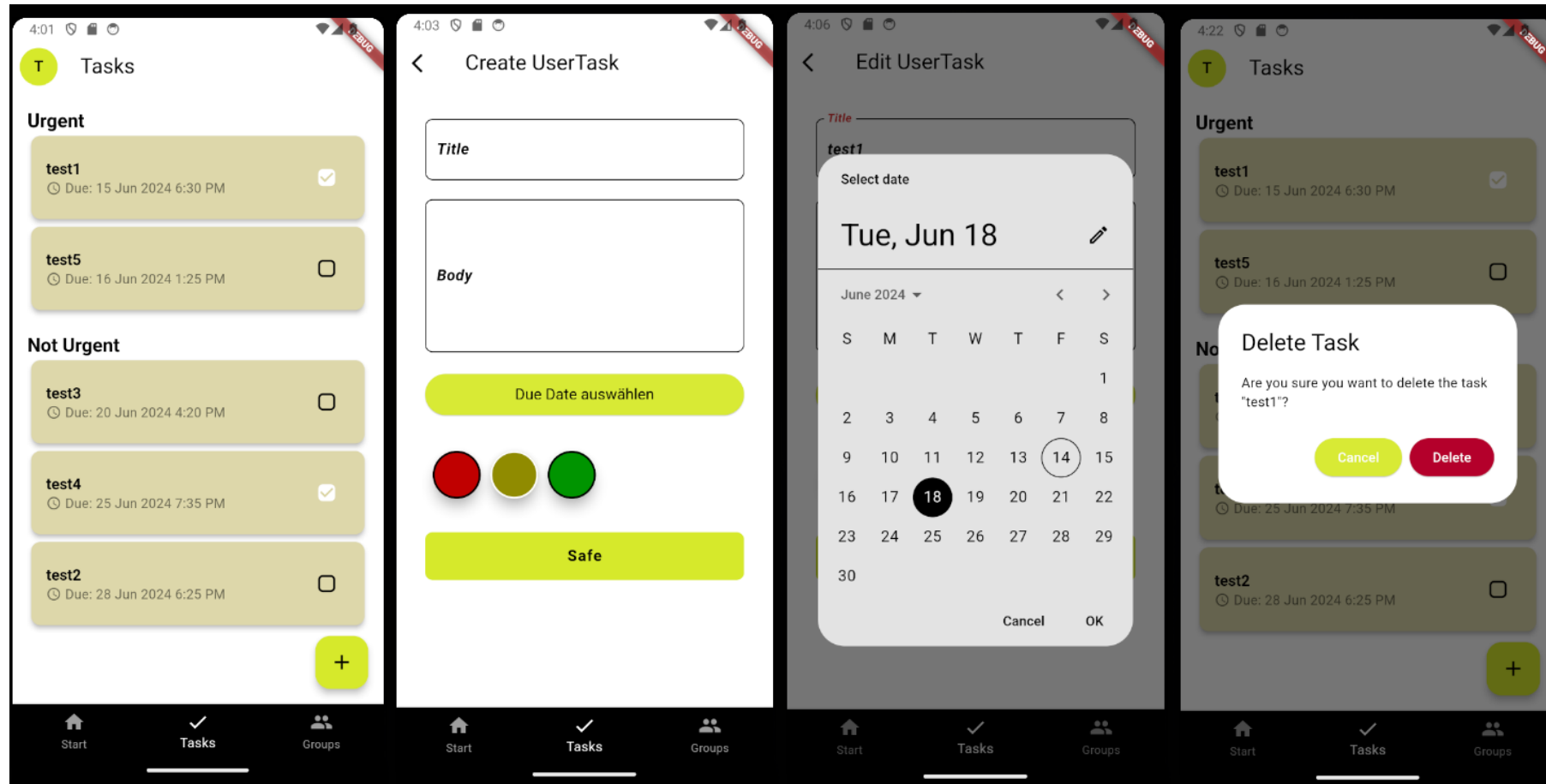
Die UI - Aufgaben

Aufgaben

Aufgabe erstellen

Aufgabe editieren
und Datum

Aufgabe löschen



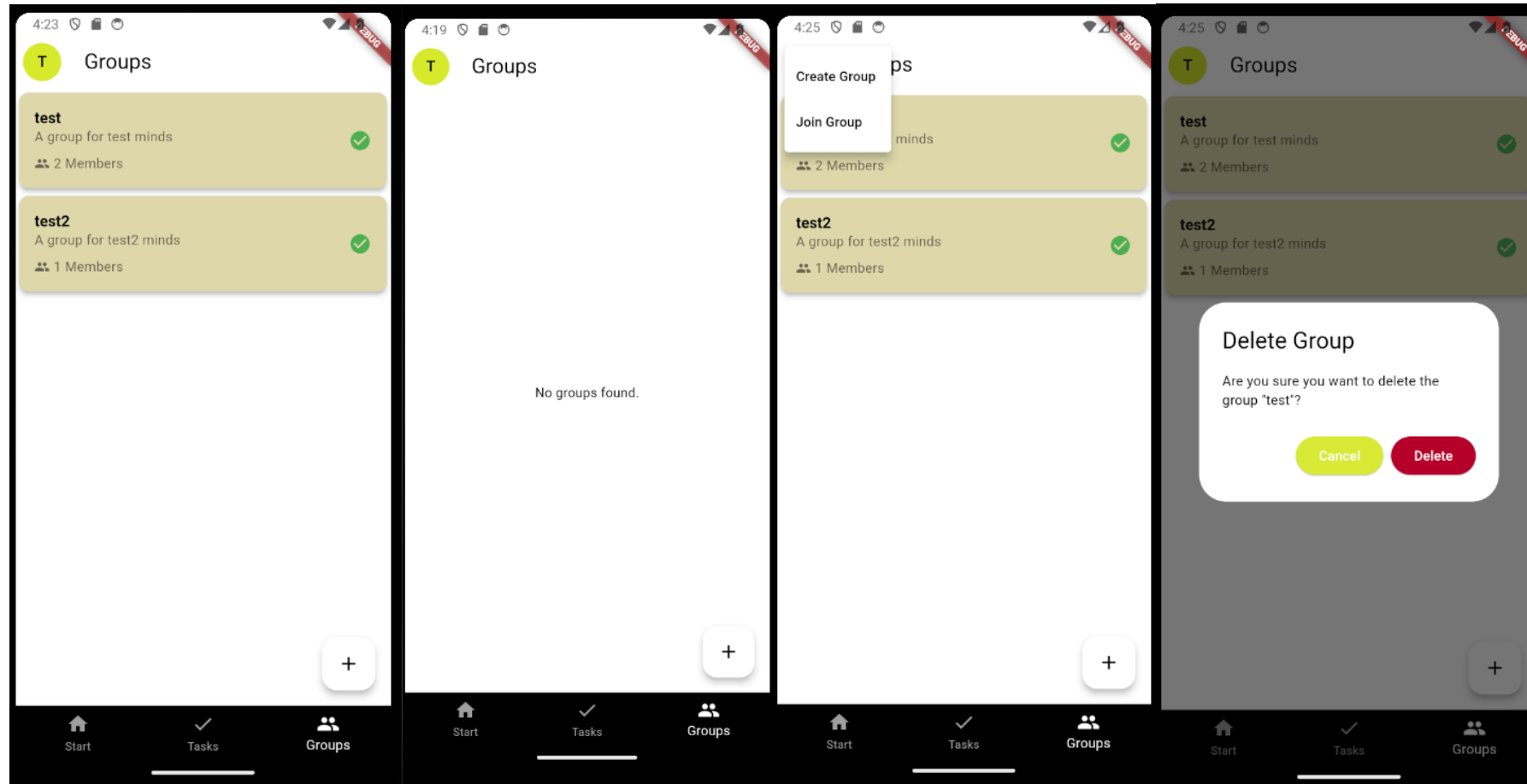
Die UI - Gruppen

Gruppen

Keine Gruppen
vorhanden

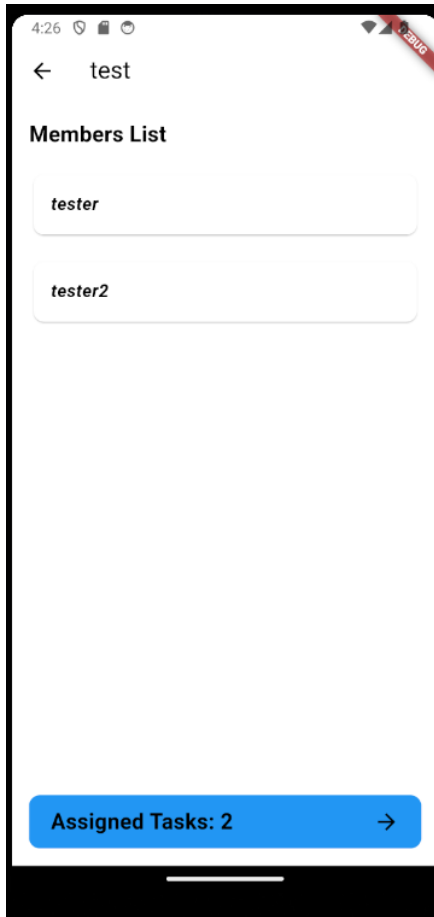
Gruppe beitreten
oder erstellen

Gruppe löschen

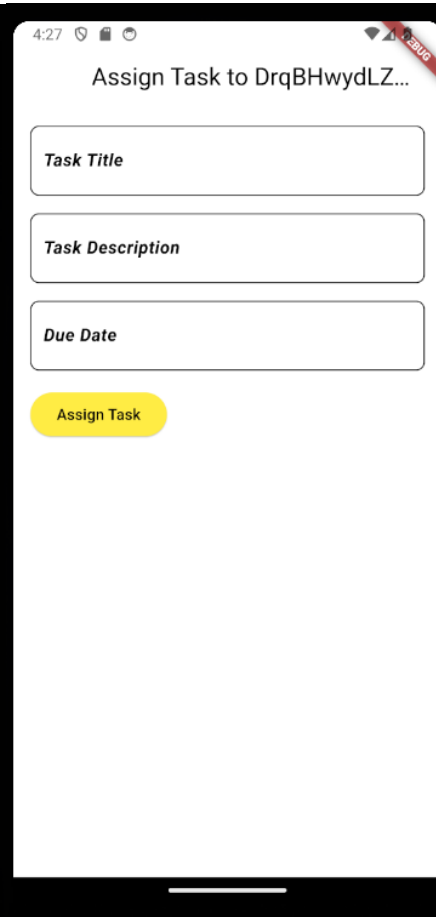


Die UI – Gruppe im Detail (noch in Bearbeitung)

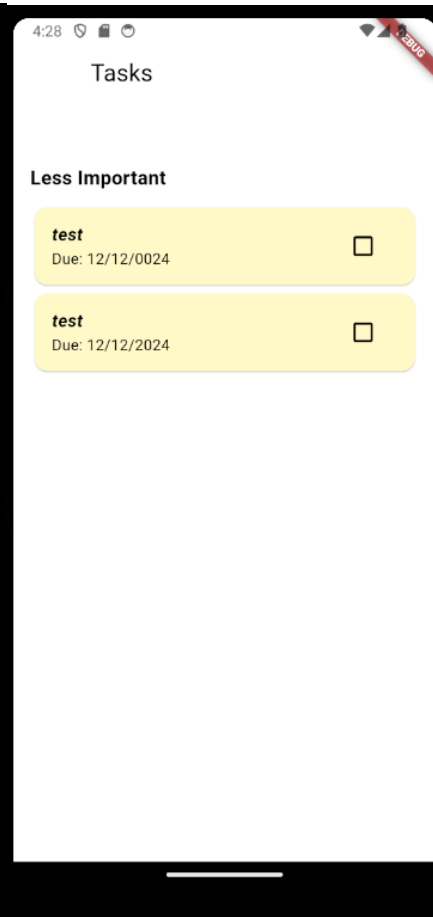
Gruppe im Detail



Aufgabe zuweisen



Zugewiesene Aufgaben



Firestore Datenbankstruktur

- Gruppen Sammlung
- Einzelne Gruppen (ID)
- Gruppe: creatorId, groupTasks(map), members, name und password
- groupTasks: assignedUserId, body, color, done, dueDate, groupId, id, serverTimeStamp, title

The screenshot displays the Google Cloud Firestore console interface. The left sidebar shows the project hierarchy: `groups` > `ISJK9P2w55Cf2`. The main panel shows the collection `groups` with a document `ISJK9P2w55Cf2CoWqVT8`. The document's structure is as follows:

```
{
  creatorId: "DrqBHwydLZQNzIbq1HRghM8bC9C3",
  groupTasks: {
    "DrqBHwydLZQNzIbq1HRghM8bC9C3": {
      0: {
        assignedUserId: "DrqBHwydLZQNzIbq1HRghM8bC9C3",
        body: "test",
        color: 4278948875,
        done: false,
        dueDate: "12. Dezember 24 um 01:05:21 UTC+1:5",
        groupId: "ISJK9P2w55Cf2CoWqVT8",
        id: "486c0d30-c84d-102a-a2c4-8fccdfde6924",
        serverTimeStamp: "14. Juni 2024 um 14:13:02 UTC+2",
        title: "test"
      },
      1: {assignedUserId: "DrqBHwyd..."}
    }
  },
  members: [
    "DrqBHwydLZQNzIbq1HRghM8bC9C3",
    "TVNzvFb9p7dYOWF5YCH3BuateWt1"
  ],
  name: "test",
  password: "test"
}
```



Ausblick

- Verbesserung der Gruppenfunktionalität:
 - Erweiterte Gruppenverwaltung (z.B. Rollen, Rechte).
 - Verbesserung der Darstellung.
- Integration von Benachrichtigungen:
 - Push-Benachrichtigungen für Gruppenaktivitäten und Aufgabenverteilung.
- Einführung einer Suchfunktion bei den Aufgaben:
 - Implementierung einer effizienten Suchfunktion für Aufgaben.



Fragen?