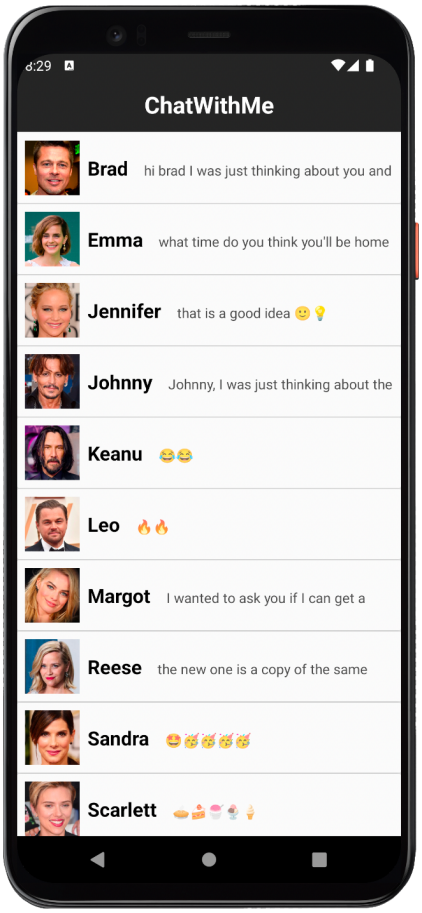
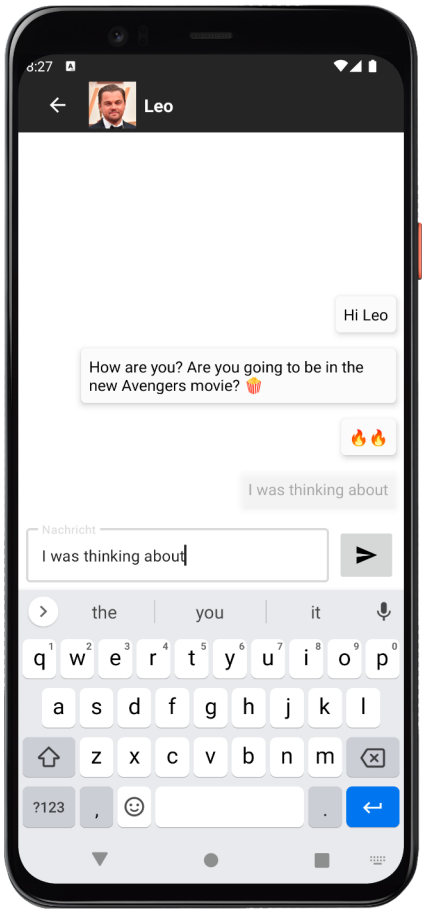
Kotlin 3.11 - Wiederholung

Aufgaben

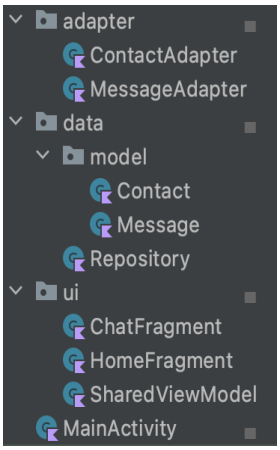
1. CHAT APP - Navigation, MVVM, LiveData

In dieser Aufgabe soll die Chat App “ChatWithMe” programmiert werden. Sie besteht aus zwei Screens, einer Ansicht mit allen Kontakten (Home Ansicht) und einer Chat Ansicht. In der Kontakte Ansicht sollen alle Kontakte mit Profilbild und Profilnamen angezeigt werden. Zusätzlich wird die zuletzt gesendete Nachricht angezeigt. Durch den Klick auf einen Kontakt in der Kontakte Ansicht wird man zur Chat Ansicht des entsprechenden Kontaktes weitergeleitet. Hier wird das richtige Bild, der richtige Name und der Chat Verlauf mit diesem Kontakt angezeigt. Dank der Einbindung von LiveData können wir außerdem ein weiteres cooles Feature einbauen: Die Anzeige eines Nachricht Entwurfs bzw. einer Draft* Message (Hier im zweiten Screenshot als ausgegraute Nachricht zu sehen). In dieser Draft* Message wird das Getippte aus dem Eingabefeld live angezeigt, sodass man sehen kann, wie die Nachricht nach dem Abschicken aussehen würde. **englisch: Draft = deutsch: Entwurf*





In der Vorlage befinden sich bereits ein simples Layout und zwei RecyclerViews, eine für die Kontakte und eine für den Chat. Die App ist folgendermaßen aufgebaut:



adapter: hier befinden sich die Adapter für die beiden RecyclerViews
- **ContactAdapter**: erstellt die Contact ViewHolder für die Kontaktliste
- **MessageAdapter**: erstellt die Message ViewHolder für den Chat

data: hier befinden sich die Informationen, die für die App vorbereitet werden
- **model**: hier befinden sich die Data Klassen, die ein Item modellieren
  - **Contact**: definiert, welche Attribute einen Kontakt ausmachen
  - **Message**: definiert, welche Attribute eine Nachricht ausmachen
- **Repository**: liefert eine Liste an Kontakten/ Nachrichten

ui: hier wird das Verhalten der App und die App Oberfläche programmiert
- **ChatFragment**: hier wird bestimmt, was die Chat-Oberfläche anzeigt
- **HomeFragment**: hier wird bestimmt, was die Home-Oberfläche anzeigt
- **SharedViewModel**: hier ist die Logik definiert, die die UI beeinflusst

MainActivity: hier wird das activity_main Layout geladen

Hinweis: Die benötigten Dependencies und Plugins sind bereits in der `build.gradle(Project)` Datei und in der `build.gradle(Module)` Datei definiert.

- [ ] Schaue dir die bereits bestehenden Klassen genau an

  Anmerkung:
  In der `SharedViewModel` Datei findest du zusätzlich zur ViewModel Klasse eine `enum` Klasse namens `DraftState`. Diese Klasse speichert nichts anderes als die Zustände, die die Draft Message annehmen kann. D.h.: Eine Draft Message kann entweder `CREATED`, `CHANGED`, `SENT` oder `DELETED` sein. Jeder dieser Zustände kann in einer Variablen dieses Typs abgespeichert werden, z.B.: `val zustand: DraftState = DraftState.CREATED`
  Später kann man dann z.B. den in der Variablen gespeicherten Zustand abfragen und in einer `if` Bedingung verarbeiten.
  Mehr Informationen zu Enum Klassen findest du hier: **Enum Classes**

- [ ] Das Repository beinhaltet bereits eine Variable `contactList`, in der die Liste an Kontakten gespeichert ist. Um auf diese Variable zugreifen zu können, müssen wir eine Instanz der Klasse Repository erstellen

  - [ ] Gehe in das ViewModel. Hier wollen wir auf die Informationen aus dem Repository zugreifen. Erstelle also eine Repository Instanz und speichere diese in einer Variablen `repository` ab.
  - [ ] Erstelle anschließend eine private, verschachtelte Variable `contactList`, in der die Liste aus Kontakten aus dem Repository gespeichert wird
  - [ ] Lass dir die Liste an Kontakten nun in der RecyclerView der Kontakt Ansicht anzeigen. Binde dafür das ViewModel in das Home Fragment ein und erstelle einen Contact Adapter für die RecyclerView `rvContacts`. Übergebe dem Adapter hierfür die Liste aus dem ViewModel
  - [ ] Um die vollständige Liste im Homescreen anzuzeigen, müssen wir die Informationen im `ContactAdaper` noch richtig zuweisen. Öffne die Klasse `ContactAdapter`

- [ ] Hole in der Funktion `onBindViewHolder` den aktuellen Kontakt aus der richtigen Position des Datasets (also aus der übergebenen Kontaktliste)
- [ ] Setze das richtige Bild und den richtigen Namen. Die Informationen dafür findest du im Kontakt gespeichert
- [ ] Prüfe, ob die `chatHistory` (Chat Verlauf) des Kontaktes Nachrichten enthält und falls ja, lass die Nachricht an oberster Stelle der Liste (Index 0) in `tvLastMessage` anzeigen
- [ ] Jetzt sollte beim Ausführen der App eine mit Kontakten gefüllte Liste im Homescreen erscheinen!

- [ ] Da die Home Ansicht nun die Kontaktliste anzeigt, ist jetzt ein guter Zeitpunkt, die Navigation zu implementieren. Bei einem Klick auf einen Kontakt soll die Chat Ansicht geöffnet werden. Ebenso soll man zurück zum Homescreen kommen, wenn man auf den Zurück-Button klickt
  - [ ] Erstelle dafür einen `nav_graph` für die Navigation und füge das Home Fragment und das Chat Fragment hinzu
  - [ ] Erstelle eine Navigationsaktion vom Home Fragment zum Chat Fragment und umgekehrt. Da wir im Chat Fragment die Information brauchen, an welcher Stelle in der Liste der angeklickte Kontakt gespeichert ist, müssen wir diese Information in der Navigation mitgeben. Füge im Chat Fragment daher noch ein Argument namens `contactIndex` vom Typ `integer` ein.
  - [ ] Wechsle zum `activity_main.xml` Layout und ersetze im `fragment` die `HomeFragment` Referenz durch die `NavHostFragment` Referenz. Füge zudem die fehlenden Navigation-Attribute hinzu
  - [ ] Um bei einem Klick auf einen Listeneintrag eine Navigation in den Chat Screen auszulösen, setzen wir einen Click Listener darauf. Programmiere diesen in Klasse `ContactAdapter`, da hier die einzelnen Listeneinträge vorbereitet werden. Übergebe hier auch die Position des Kontakts in der Liste als Argument
    Hinweis: setze den Listener auf clContact, um den gesamten Eintrag klickbar zu machen
  - [ ] Im Chat Fragment wollen wir mit einem Klick auf den Zurück-Button wieder zum Home Screen kommen, gehe daher in das Chat Fragment und setze einen Click Listener auf den Button `btnBack`. Hier soll durch die Funktion `navigateUp()` zurücknavigiert werden. Passe dafür auch die Attribute in `nav_graph.xml` an
  - [ ] Jetzt sollte die Navigation zum Chat Screen und zurück funktionieren
- [ ] Um im Chat Screen die richtigen Informationen anzuzeigen, erweitern wir jetzt das ViewModel, um darüber auf die benötigten Informationen zugreifen zu können. Öffne das `SharedViewModel`
  - [ ] hier haben wir bereits die Liste an Kontakten in einer Variablen gespeichert. Es macht Sinn, den aktuellen Kontakt, der in der Chatansicht angezeigt werden soll, in einer weiteren Variable zu speichern. Erstelle dafür eine verschachtelte Variable `currentContact`

- [ ] Erstelle zudem eine verschachtelte Variable `draftMessageState`, in der später einer der vier Zustände der Draft Message anhand des Enums gespeichert werden soll:
    - `DraftState.CREATED` : Die Draft Message wurde erstellt
    - `DraftState.CHANGED` : Die Draft Message wurde verändert
    - `DraftState.SENT` : Die Draft Message wurde abgeschickt
    - `DraftState.DELETED` : Die Draft Message wurde gelöscht

    Anmerkung:
    Die Draft Message ist ein Message Eintrag in der RecyclerView genau wie jede andere Nachricht im Chat Verlauf. Der Unterschied ist: das `isDraft` Attribut in dem Message Objekt ist `true` während es bei einer normalen Nachricht `false` sein soll. Das Erstellen einer Draft Message entspricht dem Hinzufügen eines Message Objekts an Index 0 des Chat Verlaufs, das Entfernen einer Draft Message entspricht dem Entfernen des Objekts an Stelle 0 des Chat Verlaufs. Das Senden einer Draft Message ändert nur das `isDraft` Attribut und das Ändern einer Draft Message entspricht dem veränderten Text. Vorgriff: Über alle diese Änderungen wird der Adapter der RecyclerView später im Chat Fragment benachrichtigt. Daher ist es praktisch den momentanen Zustand auslesen zu können.

- [ ] Erstelle nun noch eine Variable `inputText`. Sie ist vom Typ `MutableLiveData` und speichert einen `String` Value. Das Besondere an dieser Variable ist: Sie bekommt ihren Wert direkt LIVE aus dem XML Layout zugewiesen:

    - [ ] Gehe in das XML Layout `fragment_chat.xml`. Erstelle hier in `<data>` eine `<variable>`, die das SharedViewModel einbindet.

    - [ ] Bearbeite das `text` Attribut des `TextInputEditText` (mit der ID `textInput`), indem du das `text` Attribut mit der `inputText` Variable des ViewModel verknüpfst

      Hinweis: um die Variable zu verknüpfen, nutze die Notation “`@={...}`”

- [ ] Programmiere nun im ViewModel die Funktion `initializeChat`, sie bekommt die Position des aktuellen Kontaktes in der Kontaktliste übergeben

    - [ ] Die Funktion weist der Variable `currentContact` den aktuellen Kontakt zu. Außerdem setzt sie den Eingabe Text zurück, indem sie ihm einen leeren String zuweist. Zu Beginn soll der Draft Message Zustand `DELETED` sein, da noch keine Draft Message existieren soll.

- [ ] Programmiere die Funktion `closeChat`. Sie wird ausgeführt, wenn der Chat geschlossen wird und zurück zur Home Ansicht navigiert wird. Sie ist insbesondere wichtig, wenn eine Draft Nachricht begonnen, aber nicht abgeschickt wurde

    - [ ] Die Funktion löscht das oberste Objekt aus dem Chat Verlauf (Index 0), **aber nur**, wenn eine Draft Nachricht existiert (also Zustände `CREATED` und `CHANGED`).
      Die Funktion soll den Zustand als `DELETED` speichern

- [ ] Programmiere die Funktion `inputTextChanged`. Sie wird jedes Mal aufgerufen, wenn sich der Eingabe `text` ändert (z.B. von “Hall” zu “Hallo”), sie bekommt den `text` übergeben. In der Funktion wird von Fall zu Fall unterschieden und je nach Fall der Zustand und die `chatHistory` angepasst:
    - Falls die Draft Message existiert…
        - …und der `text` **nicht** leer ist, soll:
            - der `text` in das oberste Message Objekt (= die Draft Message) im Chat Verlauf (Index 0), in das `messageText` Attribut, gespeichert werden. Zudem soll der Zustand entsprechend in `draftMessageState` gespeichert werden, da die Draft Message geändert wurde
        - …und der `text` leer ist, soll:
            - das oberste Message Objekt (= die Draft Message) aus dem Chat Verlauf (Index 0) entfernt werden. Da wir den Draft dadurch entfernt haben, soll der neue Zustand entsprechend in `draftMessageState` gespeichert werden
    - Fall die Draft Message **nicht** existiert…
        - …und der `text` **nicht** leer ist, soll:
            - dem Chat Verlauf an oberster Stelle ein neues Message Objekt hinzugefügt werden. Das Message Objekt enthält den Eingabe `text` als `messageText` und bekommt für den `isDraft` Parameter `true` übergeben, da es sich um einen Draft handelt. Schließlich muss der Zustand in `draftMessageState` noch richtig gesetzt werden, da wir einen neuen Draft erstellt haben.

    Hinweis: Die `if` Verschachtelungen kannst du genauso aufbauen wie die Beschreibung in der Aufgabenstellung aufgebaut ist

- [ ] Programmiere die Funktion `sendDraftMessage`. Sie wird aufgerufen, wenn der User auf den “senden” Button klickt.
    - [ ] Um die Kennzeichnung der Draft Message als Entwurf zu entfernen, muss hier zuerst das `isDraft` Attribut der Message an oberster Stell des Chat Verlaufs entsprechend geändert werden. Anschließend soll noch der Zustand in `draftMessageState` richtig gesetzt werden und der Text im Eingabefeld gelöscht werden

- [ ] Da das ViewModel fertig programmiert ist und die Logik und die Zustände somit stehen, können wir uns nun abschließend darum kümmern, in der Chat Ansicht alles richtig anzuzeigen. Öffne das Chat Fragment und binde zunächst das ViewModel ein. Programmiere anschließend die Funktion `onViewCreated` :
    - [ ] Weise der `viewModel` Variable aus dem XML über `binding` das eingebundene ViewModel zu
    - [ ] Weise dem `lifecycleOwner` auf die gleiche Weise über `binding` den `lifecycleOwner` hinzu
    - [ ] Hole den bei der Navigation übergebenen `contactIndex` aus den Arguments und speichere den Wert in einer Variablen

      Hinweis: den Wert der bei der Navigation übergebenen Argumente bekommst du mithilfe der Funktion `requireArguments().getInt(key)` wobei `key` der Name des Arguments ist
    - [ ] Rufe die Funktion des ViewModel zum Initialisieren des Chats auf, wobei du den `contactIndex` übergibst
    - [ ] Setze das Profilbild und den Profilnamen anhand der Informationen, die im `currentContact` des ViewModel zu finden sind.
    - [ ] Beobachte die `inputText` Variable aus dem ViewModel und rufe bei einer Änderung die `inputTextChanged` Funktion aus dem ViewModel auf. Übergebe dabei den inputText
      Hinweis: mit `it`
    - [ ] Erstelle einen neuen Message Adapter und speichere ihn in einer Variablen ab. Übergib dabei den Chatverlauf als dataset, den du im `currentContact` des ViewModel findest. Den Kontext erhältst du mit `requireContext()`. Weise der RecyclerView anschließend den eben erstellten Adapter zu
    - [ ] Je nach Zustand der Draft Message müssen wir den Adapter über das hinzugefügte/ gelöschte/ geänderte Objekt in der Liste informieren. Beobachte daher den `draftMessageState` aus dem ViewModel, in dem der aktuelle Zustand zu finden ist.
        - Fall sich der Zustand der Draft Message zu `CREATED` ändert, soll:
            - der Message Adapter über das eingefügte Item informiert werden (die Funktion dafür lautet: `notifyItemInserted(0)`
            - die RecyclerView automatisch auf die neue Position scrollen: `binding.rvMessages.scrollToPosition(0)`
        - Fall sich der Zustand der Draft Message zu `CHANGED` ändert, soll:
            - der Message Adapter über das geänderte Item informiert werden (die Funktion dafür lautet: `notifyItemChanged(0, Unit)`
        - Fall sich der Zustand der Draft Message zu `SENT` ändert, soll:

        - der Message Adapter über das geänderte Item informiert werden (die Funktion dafür lautet: `notifyItemChanged(0, Unit)`
    - Fall sich der Zustand der Draft Message zu `DELETED` ändert, soll:
        - der Message Adapter über das gelöschte Item informiert werden (die Funktion dafür lautet: `notifyItemRemoved(0)`
- ☐ Setze einen Click Listener auf `btnSend`, in dem die entsprechende Funktion aus dem ViewModel aufgerufen wird

☐ Die Funktion `onDestroy` wird aufgerufen, wenn die Ansicht geschlossen wird und der LifeCycle des Fragments beendet wird. Rufe daher innerhalb der Funktion die ViewModel Funktion zum Schließen des Chats auf

☐ Führe die App aus und teste sie ausführlich. Wenn du willst, kannst du das relativ fade Layout anpassen und verschönern (z.B. mit Farben und mehr Features)

☐ BONUS: Setze einen `onLongClickListener` auf die CardView einer Nachricht und erstelle darin einen `shareIntent` mit dem Text der Nachricht als `EXTRA_TEXT`

Viel Erfolg! 💬