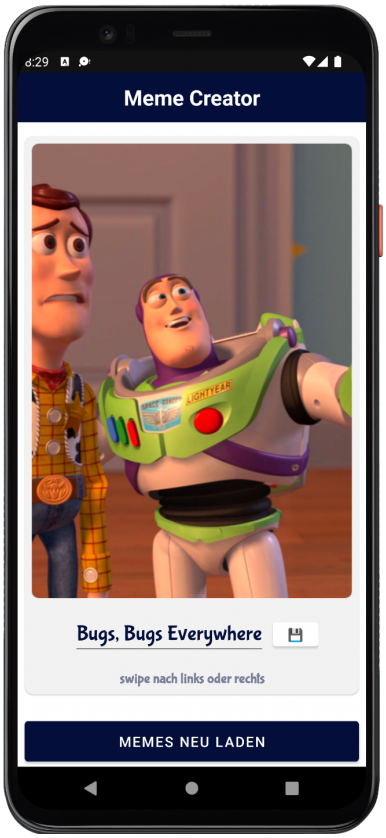
Kotlin 4.3 - API Calls und Bilder

Aufgaben

Hinweis: Zu bearbeiten ist Aufgabe 1. Aufgabe 2 ist eine Bonusaufgabe.

1. MEME CREATOR - API Call Bilder

In dieser App holen wir mit einem API-Call eine Liste aus Meme Vorlagen. Diese Memes werden wir in der App anzeigen lassen, mit der Möglichkeit, eigene Untertitel zu schreiben und zu speichern.



- Die Dokumentation der API, die wir in dieser App benutzen werden, findest Du hier: MemeAPI schau sie Dir einmal an.
- Öffne das Projekt “Meme Creator”
- Bevor es losgehen kann, müssen wir noch die richtigen Berechtigungen und Dependencies für den API-Call angeben:

  In der `AndroidManifest.xml` Datei:

```
<uses-permission android:name="android.permission.INTERNET" />
```

In der `build.gradle(Module)` Datei unter `dependencies`:

```
//Retrofit
implementation "com.squareup.retrofit2:retrofit:2.9.0"
implementation "com.squareup.retrofit2:converter-moshi:2.9.0"
implementation "com.squareup.moshi:moshi-kotlin:1.13.0"

// Coil
implementation "io.coil-kt:coil:1.1.1"
```

- Wir werden gleich den API-Call einbauen, vorher sehen wir uns noch einmal an, wie die Antwort des MemeAPI Servers aussieht:
    - Die gesamte Antwort ist in ein JSON Objekt verpackt {...}
    - Hinter der Variable “data” im JSON Objekt verbirgt sich ein weiteres JSON Objekt {...}
    - Hinter der Variable “memes” verbirgt sich wiederum ein Array [...] aus JSON Objekten
    - Jedes der JSON Objekte im Array steht für ein Meme mit mehreren Informationen, wie “name” (Titel) und “url” (Bild URL). Die JSON Meme-Objekte sind durch {...} gekennzeichnet
    - Wenn der API Call in unserer APP verarbeitet wird, wird dieser in Kotlin Objekte übersetzt, daher brauchen wir für jedes JSON Objekt eine passende Klasse, mit der man die Objekte erzeugen kann:


```
{
    "success": true,
    "data": {
        "memes": [
            {
                "id": "61579",
                "name": "One Does Not Simply",
                "url": "https://i.imgflip.com/1bij.jpg",
                "width": 568,
                "height": 335,
                "box_count": 2
            },
            {
                "id": "101470",
                "name": "Ancient Aliens",
                "url": "https://i.imgflip.com/26am.jpg",
                "width": 500,
                "height": 437,
                "box_count": 2
            }
            // probably a lot more memes here..
        ]
    }
}
```


- Erstelle zuerst eine `data` Klasse `Meme` im package `datamodels`. Diese Klasse steht für ein einzelnes Meme-Objekt und enthält Variablen, die genauso heißen wie in der Serverantwort. Wir brauchen nicht alle Informationen und daher enthält die `data` Klasse `Meme` nur zwei Variablen: `name` und `url` vom Typ `String`
- Erstelle nun eine `data` Klasse `MemeList` im package `datamodels`. Hier wird die Liste aus Meme Objekten gespeichert, die sich in der Antwort hinter “memes” verbirgt. Daher braucht die Klasse für die Übersetzung genau eine Variable `memes` vom Typ “Liste vom Typ `Meme` ”

- Erstelle nun eine `data` Klasse `MemeData` im package `datamodels`. Hier wird das JSON Objekt hinter "data" rein gespeichert. Daher braucht die Klasse für die Übersetzung genau eine Variable `data` vom Typ `MemeList`.
- Jetzt können wir den API-Service einbauen. Erstelle dazu ein **File** `MemeApiService` im package remote und programmiere sie folgendermaßen:

  Hinweis: ⇨ siehe Folien zu API Calls, Seite 13 & 14

  - Erstelle eine konstante `BASE_URL`:

    ```
    "https://api.imgflip.com/"
    ```

  - Erstelle eine private Variable `moshi`, in der ein Moshi gespeichert ist
  - Erstelle eine private Variable `retrofit`, in der ein Retrofit gespeichert ist
  - Erstelle ein Interface `MemeApiService`, in dem genau eine `suspend` Funktion zum Holen der Informationen namens `getMemes()` enthalten ist. Sie liefert den Typ `MemeData` zurück. In der `@GET` Annotation wird die URL folgendermaßen spezifiziert:

    ```
    "get_memes"
    ```

  - Erstelle schließlich noch ein `object` `MemeApi`, das eine Variable namens `retrofitService` enthält.
- Jetzt können wir den API-Service nutzen und die Informationen in der Klasse `AppRepository` abrufen, speichern und dem Rest der App zur Verfügung stellen. Gehe in die Klasse `AppRepository`
  - Füge dem Konstruktor der Klasse als Erstes eine private Variable `api` vom Typ `MemeApi` hinzu, da wir die API für den Abruf der Informationen hier brauchen
  - Erstelle eine LiveData Variable `memes`, in der alle Memes in einer Liste gespeichert werden sollen.
  - Schreibe eine `suspend` Funktion `getMemes()`, die die Informationen in einem `try catch` Block aus der `MemeAPI` holt.
    - Im `try` Teil rufst du zuerst die Funktion `getMemes()` aus dem `retrofitService` der API auf und speicherst die Antwort in einer Variablen `memeData`. Anschließend weist du der LiveData Variablen `memes` die mit `shuffled` vermischte `memes` Liste aus der Variable `data` von `memeData` zu
    - Im `catch` Block, soll mit `Log` eine eventuelle Fehlermeldung ausgegeben werden
- Jetzt sind die Informationen aus dem API-Call in der Klasse `AppRepository` zu finden. Wechsle in die Klasse `MemesViewModel`, hier können wir die Informationen jetzt verwerten
  - Erstelle ein `AppRepository` Objekt namens `repository`
  - Speichere die `memes` aus dem Repository in einer eigenen Variable namens `memes`

    - Schreibe eine Funktion `loadData()`, die dafür verantwortlich ist, den API Call über das Repository in einer Coroutine auszuführen.
- Anschließend müssen wir nur noch die erhaltenen Informationen in der UI anzeigen. Wechsle in die Klasse `MemesFragment`
    - Hier werden die API Informationen das erste Mal während LayoutAufbaus abgerufen.
    - In der Funktion `onViewCreated` können wir jetzt die Liste in der Variable `memes` aus dem ViewModel beobachten und bei jeder Änderung soll der RecyclerView `rvMemes` ein neuer `MemeAdapter` mit eben dieser Liste als Parameter zugewiesen werden
    - Damit dies funktioniert, musst du in der Klasse `MemeAdapter` noch ein paar kleine Änderungen vornehmen:
        - Ändere den Typ des `dataset` im Konstruktor von `Any` zu `Meme`
        - In der Funktion `onBindViewHolder`:
            - Hole das aktuelle `Meme` aus der `position` im `dataset` und speichere es in einer Variablen.
            - Baue eine URI aus der im Meme gespeicherten Bild URL

              Hinweis: `.toUri().buildUpon().scheme("https").build()`

            - Lade das Bild mithilfe von `load()` und der URI in die ImageView. In den geschweiften Klammer von `load()` kannst du ein `error` Bild festlegen, falls das Laden der Bilder fehlschlägt. Außerdem kannst du die Ecken des Bildes mit `transformations` abrunden
            - lade den Titel (`name`) des Memes in die TextView `tvTitle`
            - Bei einem Klick auf den Button `btnSave` soll der Text des Eingabefeldes in das Meme Objekt gespeichert werden
    - Zurück in der Klasse `MemesFragment` soll schließlich noch ein ClickListener auf den Button `btnRefresh` gesetzt werden, in dem jedes Mal ein neuer API-Call ausgeführt und die Liste aktualisiert wird.
- Die App mit API-Call sollte jetzt funktionieren! Führe sie aus und erstelle neue Memes!

Viel Erfolg! 😎

2. MEME CREATOR - Ladeanimation

Erstelle wie in der Live-App eine Ladeanimation. Sie soll sich drehen, während die Bilder geladen werden.

Viel Erfolg! 🔄