Data Detective – Audit 3

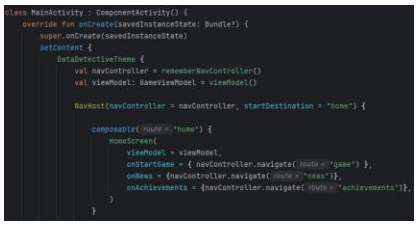
Entwicklungsprojekt WS 25/26
von Florian Roß und Karim Khemiri

Inhaltsverzeichnis

- Moderne App-Entwicklung mit Kotlin
- Überarbeitete Proof of Concepts
- Dynamische Aufgabengenerierung (PoC 101)
- Automatisch manipulierte Diagramme (PoC 103)
- Animierte Auflösung (PoC 104)
- Manipulationen (PoC 105/106)
- Rapid Prototype Life Präsentation
- Weitere Planung

Moderne App Entwicklung mit Kotlin

Klassiche Kotlin-Android Umsetzung:
- App Launch über MainActivity
- Jetpack Compose als moderne Grundlage für Android Apps
- Navigations Controller zum Wechsel von Screens
- ViewModel zur Nutzung des App internen States



Grafikbibliothek Wechsel:
Compose UI statt Kotlin Notebooks mit Kandy und Lets Plot

-> Etwas weniger Funktionsvielfalt aber für unsere Zwecke des Rapid Protype mehr als genug (PoC 102)

3

Unsere App folgt von der Architektur her komplett den neuen und modernen Standards der Android App Entwicklung die Kotlin zur Verfügung stellt. Die drei Hauptkomponenten Main Activity (zum managen der App an sich), Jetpack Compose (Zum erzeugen angeordneter UI Komponenten und dessen Interaktion) und dem View Model (für die App internen States)

Nach einigen Tagen Entwicklungszeit und Nutzungsversuchen der vorher als "gut" evaluierten Bibliotheken Kotlin Notebkooms mit Kandy und Lets Plot haben wir nach unserem PoC Fallback beschlossen stattdessen die Compose interne UI Grafik-Bibliothek zu verwenden für die Darstellung zu verwenden. Die Compose Statistiken liefen quasi direkt und legen sehr großen Wert auf Flexibilität was die Animationen auch vereinfacht hat ohne viel Kontrolle über die Darstellung zu verlieren. Andere Statistik-Bibliotheken hingegen legen starken Fokus auf eine fix festgelegte und korrekte Darstellung was sich als ein weiteres Hinderniss herausgestellt hat.

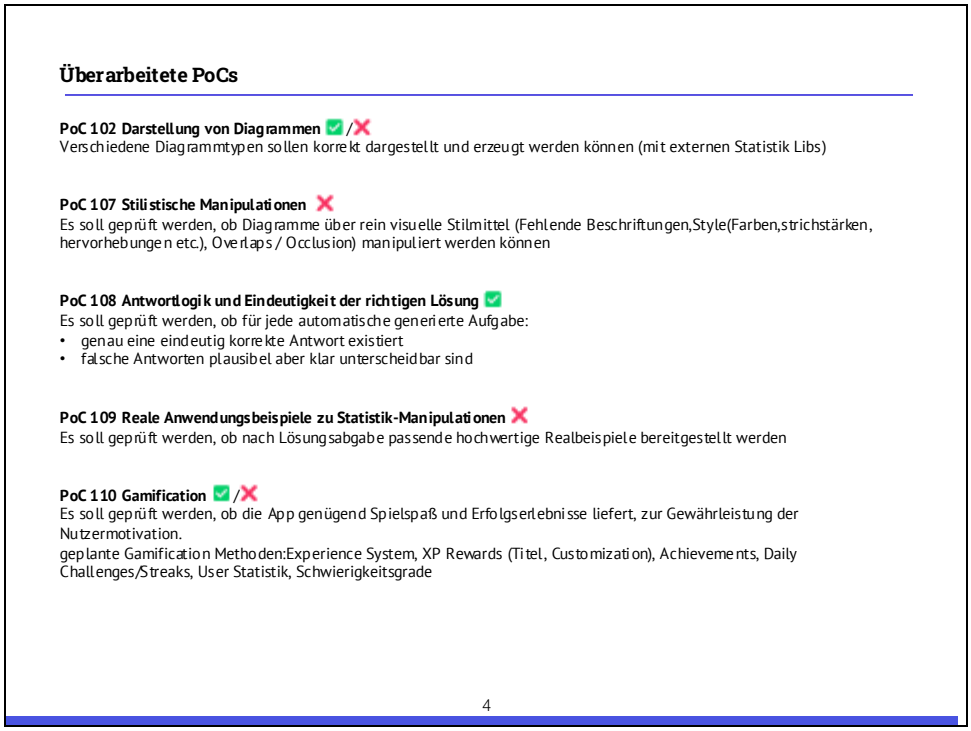
Durchgeführter Fallback bei:
"PoC 102 Darstellung von Diagrammen

(...)

Fallbacks:

1.Nutzung von Canvas-Composables anstatt externer statistiklibs

(...) "



Pocs wurden nocheinmal überarbeitet.
Auf 1,3,4,5,6 wird in den nächsten folien eingegangen

POC102 Darstellungen von Diagrammen: Linien und Balkebdiagramm(horizontal/vertikal) wurden umgesetzt
PoC107 Stilistische Manipulationen

fehlt noch
Poc108 Antwortlogik umgesetzt
Poc109 Logik für "Where was this used?"Button fehlt noch
Poc110 Gamification fehlt auchnoch größtenteils

PoC 1: Dynamische Aufgabengenerierung

PoC 101 Dynamische Generierung von Statistikaufgaben
Beschreibung des Vorhabens:
Statistikaufgaben sollen automatisch und konsistent aus Basis-Datensätzen erzeugt werden.
(...)

Umsetzung:
TaskGenerator.genrate(...)
Zufällige Auswahl:
- Datensatz
- Kompatibler Diagrammtyp
- Kompatibler Manipulationstyp
- Stärke der Manipulatonsparameter(sinnvoll begrenzt)
- Antwortmöglichkeiten(mit genau einer richtgen)

Ergebnis:
Vollständige Aufgabe die an ManipulatedChart übergeben werden kann

```
object TaskGenerator {
    2 Usages
    fun generate(base: ChartData): Task {
        val chartType = base.supportedChartTypes.random()      //Zufälliger Diagrammtyp aus erlaubten Typen
        val allowedManipulations = chartType.allowedManipulations          //Erlaubte Manipulationen für den Diagrammtyp
        val manipulationType = allowedManipulations.random()          //Zufällige Manipulation aus den erlaubten
        val random = Random( seed = base.id)
        //Erzeugt ein Manipulations-Objekt, das beschreibt, wie das Diagramm manipuliert wird
        val manipulation = Manipulation(type = manipulationType, //ManipulationsTyp
            intensity = ManipulationParameters.manipulationIntensity(manipulationType, random), //zufällige Stärke der Manipulation
            categoryRange = if (manipulationType == ManipulationType.TRUNCATED_CATEGORY_AXIS)
                ManipulationParameters.categoryRange(base.xData.size, random)//erzeugt bei Kategorie-Achse Manipulation einen zufälligen Bereich von Kategorien
            else null)
        val correctAnswer = AnswerPool.correctAnswerFor(manipulationType) //Richtige Antwort basierend auf Manipulationstyp
        val (options, correctIndex) = generateAnswerOptions(correctAnswer,  pool = AnswerPool.allAnswers)//Erzeugt Antwortmöglichkeiten
        return Task(
```


Über den Taskgenerator wird eine Aufgabe generiert.
Der Taskgenerator bekommt einen zufälligen Datensatz(enthält Titel, Rohdaten, Kategorien, Einheit, Beschreibung, kompatible Diagrammtypen)
Für diesen wird ein zufälliger kompatibler Diagrammtyp ausgewählt
Für Datensatz und Diagrammtyp wird eine zufällige kompatible

Manipulationsmethode gewählt
Zudem werden noch Manipulationsparameter für die jeweilige Manipulation generiert und zufällige Antwortmöglichkeiten(+ richtige)
Ausgeben wird dann ein Task welcher die Werte vom Datensatz, Diagrammtyp, Manipualtionstyp und -parameter und Antwortmöglichkeiten enthält
Dieser Task kann gerendert werden (manipuliert/korrekt)

(...)
Der Aufgabengenerator soll auf Basis einen zufälligen Datensatzes:

- Einen zufälligen (aber kompatiblen) diagrammtypen wählen
- Einen zufälligen (aber kompatiblen) Manipulationstypen wählen

- Zufällige (aber sinnvoll begrenzte) Stärke der Manipulationsparameter generieren
- Zufällige Antwortmöglichkeiten(mit passender richtiger Antwort) generieren

EXIT:

- Aufgaben werden ohne Fehler erzeugt
- Jede Aufgabe enthält genau eine korreke Antwort
- Ungültige Kombinationen(von Diagrammtyp/Manipulationstyp/Datensatz) werden ausgeschlossen
- Jede erzeugte Aufgabe ist vollständig(Diagrammdaten, Manipulationsdaten, Antworten)

FAIL:

- leere oder fehlerhafte Aufgaben
- fehlerhafte Antwortmöglichkeiten

Fallbacks:

- Reduktion der Zufälligkeit
- Einschränkung auf feste Zuordnung von Manipulationstyp für Diagrammtyp

- Vorrübergehend statische Aufgaben

PoC 3: Automatisch manipulierte Diagramme

PoC 103 Generierung von manipulierten Diagrammen
Beschreibung des Vorhabens
Es soll geprüft werden, ob manipulierte Diagramme automatisch aus unveränderten Basisdatensätzen erzeugt werden können
Das System soll

- ein unverändertes Basisdiagramm als Ausgangspunkt verwenden
- abhängig vom Manipulationstyp und Manipulationsstärkeparameter Diagrammmerkmale dynamisch verändern
- die Manipulation rein über Darstellungslogik umsetzen, nicht durch veränderung der Rohdaten (...)

Umsetzung:
ManipulatedChart() bekommt eine Task übergeben und berechnet daraus die manipulierten Chart-Werte.

Aus diesen Werten wird dann das jeweilige diagramm gerendert.

```
//Rendering je nach ChartTyp
when (task.chartType) {
    ChartType.BAR -> {
        BarChart(...) }
    ChartType.LINE -> {
        LineChart(...) }
    ChartType.HORIZONTAL_BAR -> {...}
}
```

```
@Composable
fun ManipulatedChart(task: Task, showCorrect: Boolean, modifier: Modifier = Modifier
    .fillMaxWidth().height( height = 260.dp)) {
    val ydata = task.yValues
    val xData = task.xData
    val unit = task.unit
    val manipulation = task.manipulation
    //unmanipulierte y-Werte
    val rawMinY = 0f
    val rawMaxY = ydata.maxOrNull() ?: 1f
    //Zielwerte für Werte-Achse
    val targetMinY = if (showCorrect) rawMinY
                     else manipulation.type.manipulateMinValue(ydata, manipulation)

    val targetMaxY = if (showCorrect) rawMaxY
                     else manipulation.type.manipulateMaxValue(ydata, manipulation)
    //Gesamtanzahl der Kategorien
    val totalCount = xData.size
    //Startpunkt des sichbaren Kategoriebereich (wird nur verändert bei TRUNCATED_CATEGORY_AXIS Manipulation)
    val targetCategoryStart = if (showCorrect || manipulation?.categoryRange == null) { 0f }
                              else { manipulation.categoryRange.first.toFloat() }
    //Endpunkt des sichbaren Kategoriebereich (wird nur verändert bei TRUNCATED_CATEGORY_AXIS Manipulation)
    val targetCategoryEnd = if (showCorrect || manipulation?.categoryRange == null) { (totalCount - 1).toFloat() }
                            else { manipulation.categoryRange.last.toFloat() }
```


Vorgehen:

System rendert aus einem Task ein korrektes Diagramm berechnet die Manipulationsparameter und wendet diese anschließend nur auf die Darstellung an (so kann über showCorrect manipuliert/korrekt animiert werden)

(...)

EXIT:

- Manipulierte Diagramm werden automatisch aus denselben Rohdaten erzeugt
- Jede Manipulation ist visuell eindeutig erkennbar
- Manipulation kann jederzeit rückgängig gemacht werden

FAIL:

- Manipulation erfordert manuelle Anpassung der Daten
- Manipulation wirkt zufällig oder nicht nachvollziehbar

Fallbacks:

- Vorab berechnete Manipulationswerte statt vollständig dynamischer

- Berechnung
- Einschränkung der Manipulationstypen
- Statische Beispielmanipulationen

PoC 4: Animierte Auflösung

PoC 104 Animierte Auflösung von Manipulationen
Beschreibung des Vorhabens:
Es soll geprüft werden, ob statistische Manipulationen animiert aufgelöst werden können. Dabei soll zwischen manipulierten und korrekten Zustand gewechselt werden, ohne das Diagramm neu zu laden (...)

Umsetzung:
In ManipulatedChart kann durch showCorrect(boolean) zwischen manipuliertem und korrektem Zustand gewechselt werden.

Über animateFloatAsState wird dieser Wechsel animiert.

```
//Animationen für showCorrect wechsel
val animatedMinY by animateFloatAsState(
    targetValue = targetMinY,
    animationSpec = tween( durationMillis = 1200, easing = FastOutSlowInEasing))

val animatedMaxY by animateFloatAsState(
    targetValue = targetMaxY,
    animationSpec = tween( durationMillis = 1200, easing = FastOutSlowInEasing))

val animatedCategoryStart by animateFloatAsState(
    targetValue = targetCategoryStart,
    animationSpec = tween( durationMillis = 1200, easing = FastOutSlowInEasing))

val animatedCategoryEnd by animateFloatAsState(
    targetValue = targetCategoryEnd,
    animationSpec = tween( durationMillis = 1200, easing = FastOutSlowInEasing))
```


In ManipulatedChart kann durch showCorrect(boolean) zwischen manipuliertem und korrektem Zustand gewechselt werden.
Über animateFloatAsState wird dieser Wechsel animiert.

(...)

EXIT:

- Diagrammveränderungen werden

- sichtbar animiert
- Übergang zwischen Manipuliert und korrekt ist flüssig

FAIL:

- sprunghafterwechsel ohne animation
- Diagramm wird falsch oder garnicht transformiert

Fallbacks:

- Animationen einschränken
- Statischer Wechsel zwischen manipuliert und korrekt

PoC 5/6: Manipulationen

PoC 105 Werteachsen Manipulationen :
Bei Werteachsen Manipulation wird die Werteachse angehoben (also z.B. Startpunkt nicht bei 0)
PoC 106 Kategorieachsen Manipulationen :
Bei Kategorieachsen Manipulation wird der Kategoriebereich verringert(also z.B. weniger Balken)

```
enum class ManipulationType {
    5 Usages
    TRUNCATED_VALUE_AXIS { //Verkürzte Werte-Achse (Y bei Bar/Line, X bei HorizontalBar) damit Achse nicht bei 0 beginnt
        override fun manipulateMinValue(data: List<Float>, manipulation: Manipulation): Float {
            val min = data.minOrNull() ?: 0f
            return min * manipulation.intensity } //Minimalwert wird auf einen Wert über 0 gesetzt
    },
    5 Usages
    TRUNCATED_CATEGORY_AXIS { //Verkürzte Kategorie-Achse(X bei Bar, Y bei HorizontalBar)
        override fun categoryRange(size: Int, manipulation: Manipulation): IntRange? =
            manipulation.categoryRange //berechnet eingegrenzten Kategorie Bereich
    },
```

Über manipulation.intensity wird gesichert, das die Manipulation immer unterschiedlich stark ist.

```
fun manipulationIntensity(type: ManipulationType, random: Random): Float =
    when (type) {
        ManipulationType.TRUNCATED_VALUE_AXIS -> random.nextFloat() * 0.13f + 0.80f // Achse beginnt bei 80-98% des Minimalwerts
```

Über manipulation.categoryRange wird gesichert, dass die Größe des Kategoriebereichs immer zufällig ist.

```
fun categoryRange(size: Int, random: Random): IntRange {
    //Wert für Sichtbaren Ausschnitt für 40-70 % der Kategorien
    val windowSize = (size * (random.nextFloat() * 0.3f + 0.4f)).toInt().coerceAtLeast( minimumValue = 2)
    // Zufälliger Startpunkt
    val start = random.nextInt( from = 0,  until = size - windowSize + 1)

    return start .. until < (start + windowSize)
}
```


Es wurden bisher zwei Manipulationsmechanismen umgesetzt
Werteachsen Manipulaltion, hier wird die Achse der Werte verändert(bei Liniendiagramm und normalem Balkendiagramm = y-Achse, bei Horizontalen Balkendiagramm = x-Achse), damit kann z.B. keine 0-Baseline

umgesetzt werden oder auch eine allgemeine Verkürzung/Verlängerung der Achse
Kategorieachsen Manipulation (funktioniert nicht bei LinienDia), hier wird der Kategoriebereich verkleinert (also weniger Balken (bei horziontal auf der y-Achse, bei vertikal auf der x-Achse))
Um die Zufälligkeit der Manipulationen zu gewährleisten werden zur jeweiligen Manipulation Parameter generiert, die zufällig aber sinnvoll begrenzt sind (bei z.B. Werteachsenmanipulation wird der achsenbeginn auf 80-98% des Minmalwert gestellt anstatt auf 0, damit die Manipulation auffälig genug ist und dennoch jedes mal zufällig intensiv ist) (bei z.B.

Kategorieachsenmanipulatonen wird der sichtbare Kategoriebereich auf 40-70% gestellt(mit zufälligem Startpunkt)

PoC 105 Werteachsen Manipulationen
Beschreibung des Vorhabens:
Es soll geprüft werden, ob numerische Werteachsen gezielt manipuliert werden können(durch verkürzten Startpunkt der Achse(also Startpunkt über 0) oder veränderte Skalierung der Achse)
Die Manipulation soll unabhängig vom Diagrammtyp funktionieren(z.B y-achse bei vertikalBalkenDia/LinienDia, x-achse bei HorizontalBalkenDia)
EXIT:

- Werteachsen können gezielt verkürzt werden

- Manipulation ist visuell eindeutig erkennbar

FAIL:

- Werteachsen werden falsch oder gar nicht skaliert
- Manipulation ist visuell nicht eindeutig erkennbar

Fallbacks:

- Wertachsen-manipulation nur bei vertikalBalkenDiagramm

PoC 106 Kategorieachsen Manipulationen

Beschreibung des Vorhabens:

Es soll geprüft werden, ob Kategorieachsen(bei Balkendiagramm z.b die Anzahl der Balken) manipuliert werden können, indem nur ein Teilbereich der Kategorien angezeigt wird.(Auswahl eines günstigens Zeitfensters oder ausblenden

früherer oder späterer Datenpunkte)

EXIT:

- Nur ein Teilbereich der Kategorien wird dargestellt
- Achsenbeschriftungen passen zum sichtbaren Bereich
- Übergang zwischen Teilbereich und vollständiger Ansicht ist möglich

FAIL:

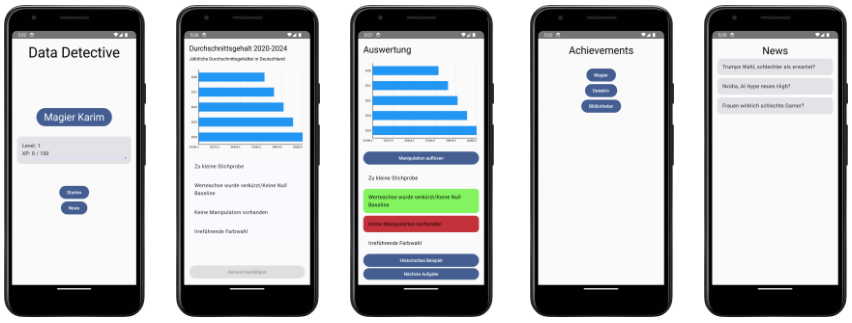
- Kategoriepunkte werden falsch dargestellt/positioniert)
- Kategorien und Daten stimmen nicht überein
- Falsche oder Fehlende beschriftung

Fallbacks:

- Statische Auswahl eines festen Kategorieausschnitts

Data Detective Rapid Protype

(Placeholder für den Foliensatz)



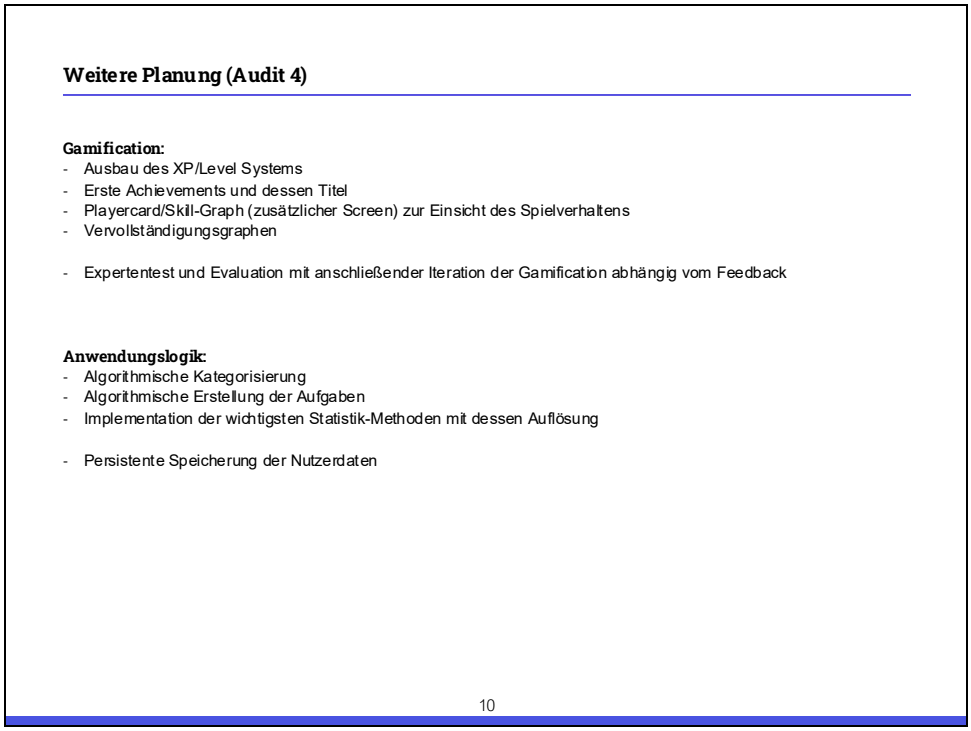
Main **Aufgabe** **Auflösung** **Achievements (Titel)** **News**

9

Die App besitzt bereits 5 funktionale Screens:

- Main-Screen: Wird beim Start der App geöffnet und dient als Navigationspunkt für alle anderen Screens
- Aufgaben-Screen: Geöffnet durch den "Starten" Button, generiert automatisch eine Aufgabe aus unseren Sample Daten und launched diese als

- Frage
- Auflösung-Screen: Nach Auswahl einer Lösung kann transferiert die Frage vom Aufgaben in den Auflösungsscreen in dem der Nutzer die die eigene und die richtige Antwort sehen kann, außerdem öffnet die Auflösung Extra Button mit dem zwischen der normalen und beeinflussten Statistik hin und her gewechselt werden können, einen Button "historisches Beispiel" der zu einem Realbeispiel verweißt in dem dieses gezeigt und erklärt wird und dem Button "Nächste Aufgabe" mit dem eine neue Aufgabe gestellt werden kann.



Da der Rapid Prototype bereits viele Grundkonzepte des geplanten Systems eingeschränkt umsetzen konnte, ist der Fokus für den weiteren Verlauf des Projekts die fehlenden nach den Prioritäten vertieft ins System zu integrieren. Die für unser Projekt sehr wichtige Gamification ist derzeit nur in sehr simplem Level System vorhanden und die evaluierten Motivatoren sind noch

nicht vorhanden. Die Nutzertests wurden zur schnelleren Entwicklung durch einen Experten-Test und ein Kompetentgespräch ersetzt.

Die Anwendungslogik ist derzeit nur in sehr einfacher Form vorhanden und unterstüzt nur 4 verschiedene Aufgabentypen. Dies soll nach dem Schema der Statistik-Methoden Tabelle aus Audit 3 vervollständigt werden. Dabei benötigt das Hinzufügen neuer Methoden auch gleichzeitig dessen dynamische Erstellung und die Auswertung welche Datensätze diese Methodik unterstüzten.

Anschließend sollen die Daten des Nutzers auch Persistenz gespeichert werden damit in den internenen Nutzerstatistiken Tendenzen angezeigt werden können und Achievements

und dessen gesetzte Titel weiter für Nutzermotivation sorgen.