Zusammenfassung des Quellcode-Refactorings – Clean-Code-Praxis

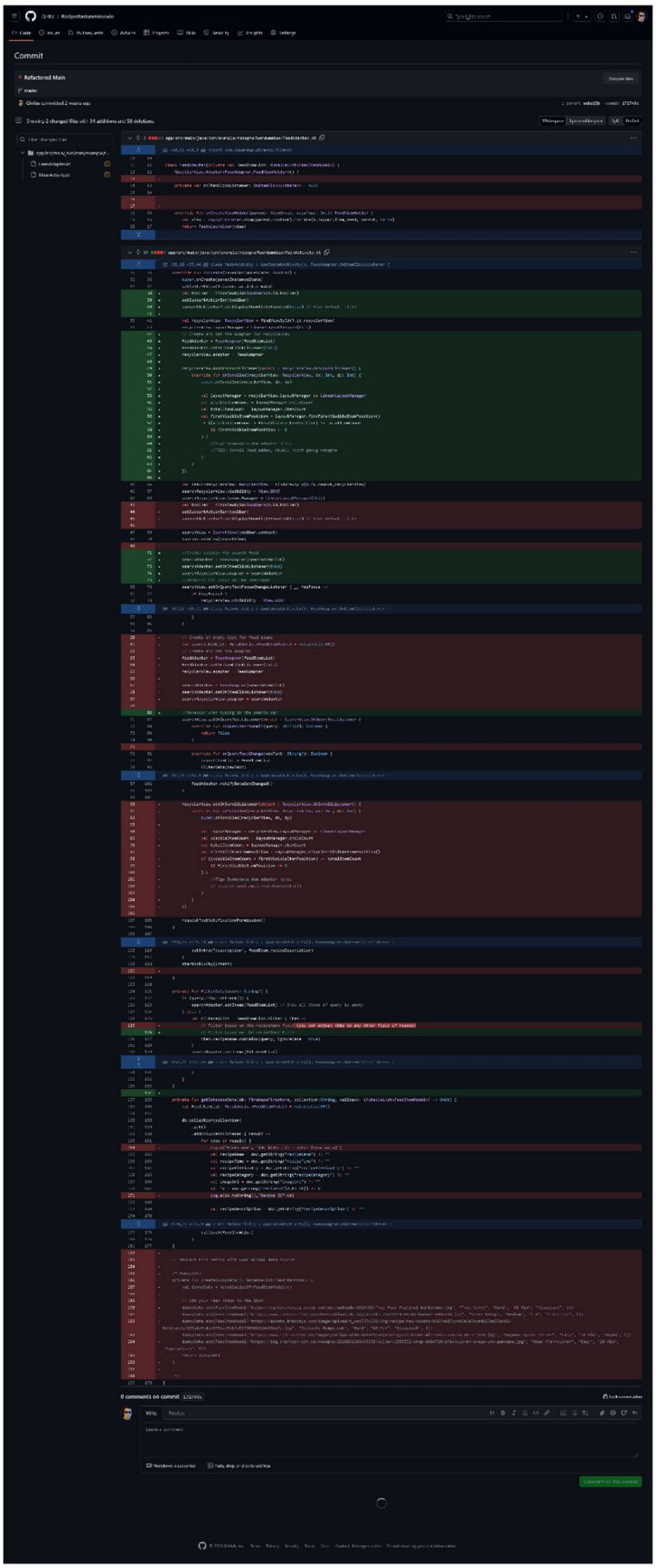
Einheitliche Struktur: Die konsistente Benennung der Variablen stellt sicher, dass der Code eine einheitliche Struktur aufweist, was wiederum die Wartbarkeit verbessert.

Entfernung von Redundanzen: Durch das Entfernen überflüssiger Leerzeichen wird der Code sauberer und konsistenter gestaltet, was auch die Lesbarkeit fördert.

Bessere Wartbarkeit: Die durchgeführten Refaktorisierungen zielen darauf ab, den Code wartbarer zu machen, indem sie eine klare Struktur schaffen und sicherstellen, dass zukünftige Änderungen leichter durchgeführt werden können.

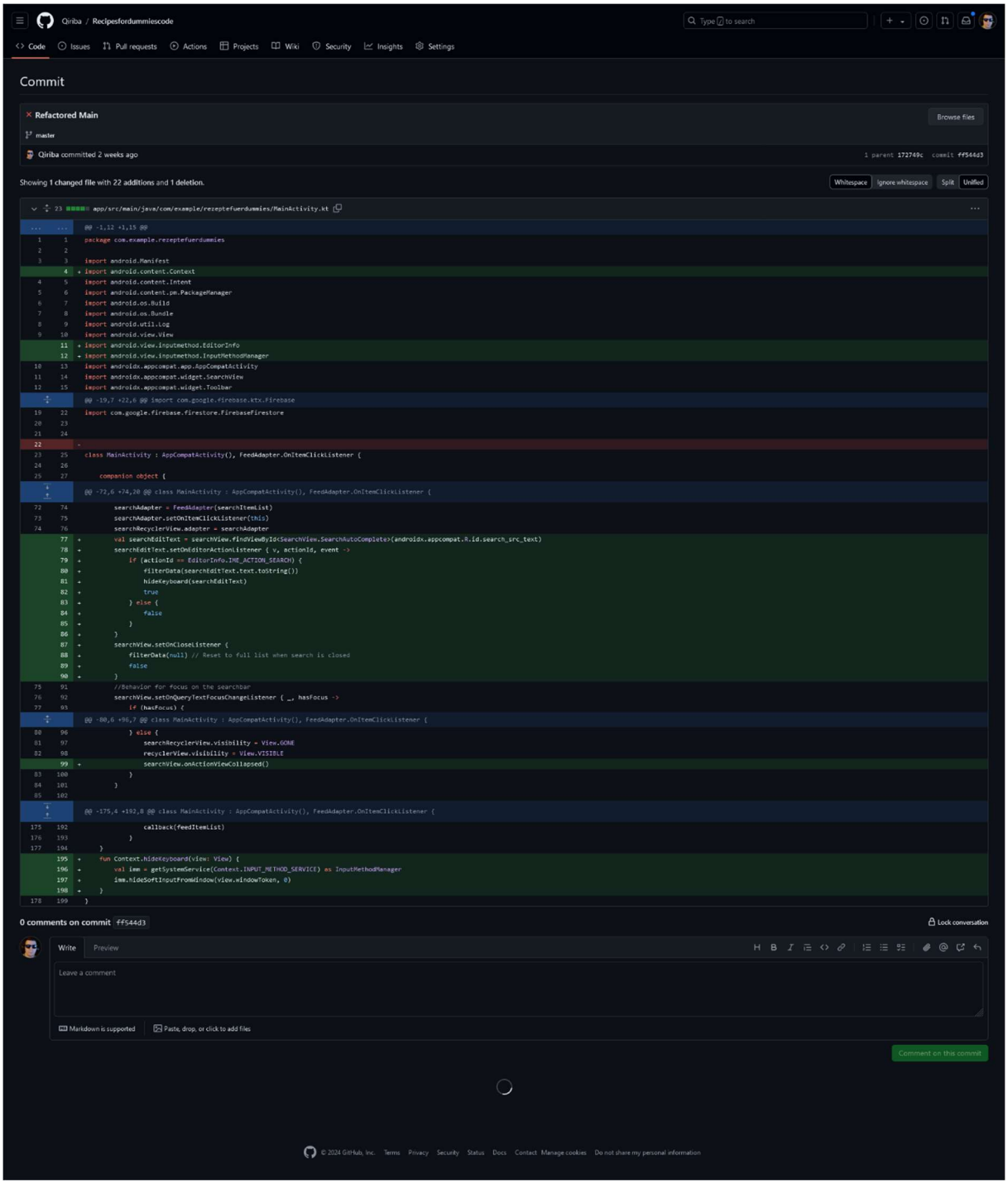
Link zu m Commit

https://github.com/Qiriba/Recipesfordummiescode/commit/172749cd63a29361d31d715c22147b59cd56ab18



Link zum Commit

https://github.com/Qiriba/Recipesfordummiescode/commit/ff544d31b3f32ae00198286338541d6e112f3b2c



Review-Protokoll

Review 1

Tobias, Marcel und Luca haben ein Review-Meeting durchgeführt. Dabei wurde sich auf die MainActivity.kt, die Hauptseite, konzentriert.

Datum: 26.05.2024 14 Uhr

Teilnehmer: Luca Knaus, Tobias Maaßen (Dokumentieren), Marcel Rohde (Zeitwächter)

Ziel/Schwerpunkt: Evaluation der Skalierbarkeit und Wartbarkeit

Komponenten für den Review: Hauptseite (MainActivity.kt)

Kriterien für den Review: Codequalität, Skalierbarkeit und Wartbarkeit

Review-Methodik: Walkthrough

Ergebnisse:

- Es ist Skalierbar, jedoch abhängig von der Performance des Servers
- Das anzeigen der Rezepte ist abhängig von der Performance der Datenbank
- Codequalität war unbefriedigend, diese wurde anschließend verbessert

Review 2

Tobias, Marcel und Luca haben ein Review-Meeting durchgeführt. Dabei wurde sich auf die MainActivity.kt, die Hauptseite, konzentriert.

Datum: 19.06.2024 17 Uhr

Teilnehmer: Luca Knaus, Tobias Maaßen (Dokumentieren), Marcel Rohde (Zeitwächter)

Ziel/Schwerpunkt: Codequalität verbessern, Refactoring der Codebasis

Komponenten für den Review: Hauptseite (MainActivity.kt), neue Hauptseite (HomeFragment.kt)

Kriterien für den Review: Codequalität und Wartbarkeit

Review-Methodik: Walkthrough und Codereview

Ergebnisse:

- Die Hauptseite war unlogisch sortiert
- Zu viele Abhängigkeiten
- Keine Kapselung der Services

Testbericht

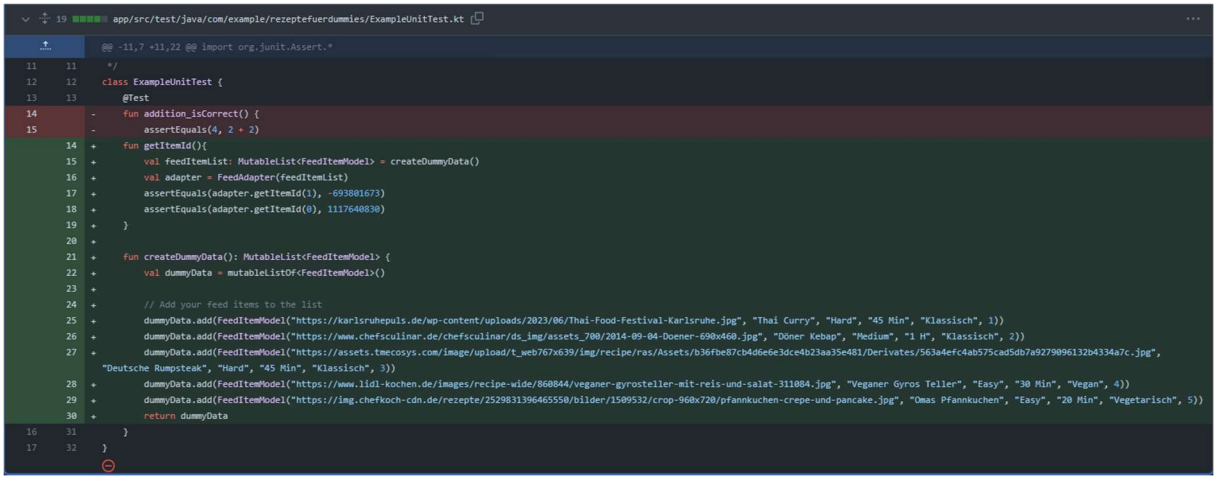
Testplan

- Wir verwenden das JUnit-Test-Framework für unsere Unit Tests. Derzeit haben wir keine Pläne für instrumentierte Tests, aber wir planen, später UI-Tests durchzuführen. Aktuell gibt es dafür noch nicht genügend testbare UI-Elemente
- Wir wollen eine Unit Test Abdeckung von ca. 60% erreichen.
- Als automatisierte Testwerkzeuge benutzen wir GitHub Actions.
- Zur Verfolgung der Testergebnisse können wir in Android Studio den Testbericht anzeigen, der nach dem Ausführen der Tests generiert wird. Dieser Bericht zeigt an, welche Tests bestanden und welche fehlgeschlagen sind.

Unit Test

Bei der Erstellung des Projektes sind Unit Tests standartmäßig bereits erstellt worden.

Hier ist ein Implementierter Test in unserem Code



app/src/test/java/com/example/rezeptefuerdummies/ExampleUnitTest.kt

```
@@ -11,7 +11,22 @@ import org.junit.Assert.*
 */
class ExampleUnitTest {
    @Test
-   fun addition_isCorrect() {
-       assertEquals(4, 2 + 2)
+   fun getItemId(){
+       val feedItemList: MutableList<FeedItemModel> = createDummyData()
+       val adapter = FeedAdapter(feedItemList)
+       assertEquals(adapter.getItemId(1), -693801673)
+       assertEquals(adapter.getItemId(0), 1117640830)
+   }
+
+   fun createDummyData(): MutableList<FeedItemModel> {
+       val dummyData = mutableListOf<FeedItemModel>()
+
+       // Add your feed items to the list
+       dummyData.add(FeedItemModel("https://karlsruhepuls.de/wp-content/uploads/2023/06/Thai-Food-Festival-Karlsruhe.jpg", "Thai Curry", "Hard", "45 Min", "Klassisch", 1))
+       dummyData.add(FeedItemModel("https://www.chefsculinar.de/chefsculinar/ds_img/assets_700/2014-09-04-Doener-690x460.jpg", "Döner Kebap", "Medium", "1 H", "Klassisch", 2))
+       dummyData.add(FeedItemModel("https://assets.tmecosys.com/image/upload/t_web767x639/img/recipe/ras/Assets/b36fbe87cb4d6e6e3dce4b23aa35e481/Derivates/563a4efc4ab575cad5db7a9279096132b4334a7c.jpg",
+ "Deutsche Rumpsteak", "Hard", "45 Min", "Klassisch", 3))
+       dummyData.add(FeedItemModel("https://www.lidl-kochen.de/images/recipe-wide/860844/veganer-gyrosteller-mit-reis-und-salat-311084.jpg", "Veganer Gyros Teller", "Easy", "30 Min", "Vegan", 4))
+       dummyData.add(FeedItemModel("https://img.chefkoch-cdn.de/rezepte/2529831396465550/bilder/1509532/crop-960x720/pfannkuchen-crepe-und-pancake.jpg", "Omas Pfannkuchen", "Easy", "20 Min", "Vegetarisch", 5))
+       return dummyData
    }
}
```

Software-Metriken – die Messwerte

1. Mean Time Between Failures (MTBF)

MTBF ist eine wichtige Zuverlässigkeitsmetrik, die die durchschnittliche Zeit zwischen dem Auftreten von Fehlern in unserer Software misst. Sie gibt uns wertvolle Einblicke in die Stabilität und Zuverlässigkeit unseres Systems. Die Berechnung erfolgt wie folgt:

$$\text{MTBF} = \frac{\text{Gesamtlaufzeit}}{\text{Anzahl der Fehler}}$$

Durch die Überwachung der MTBF können wir erkennen, wie oft Fehler auftreten und welche Maßnahmen erforderlich sind, um die Systemstabilität zu verbessern.

2. Effektivität der Tests

Diese Metrik misst, wie gut unsere Tests Fehler in der Software aufdecken. Sie berücksichtigt die Anzahl der gefundenen Fehler und wie kritisch diese sind. Eine hohe Effektivität der Tests zeigt, dass unsere Teststrategie erfolgreich ist und die wichtigsten Probleme frühzeitig identifiziert werden. Wir berechnen die Effektivität der Tests anhand folgender Formel:

$$\text{Effektivität der Tests} = \frac{\text{Anzahl der gefundenen Fehler}}{\text{Gesamtzahl der getesteten Funktionalitäten}}$$

Indem wir die Effektivität der Tests messen, können wir sicherstellen, dass unser Testprozess robust und umfassend ist.

3. Anzahl der offenen Fehler

Die Anzahl der offenen Fehler ist eine einfache, aber wichtige Metrik, die angibt, wie viele bekannte Probleme oder Fehler im Projekt noch nicht behoben wurden. Die Berechnung ist sehr direkt: Sie zählen einfach die Anzahl der offenen Fehler in Ihrem Fehlerverfolgungssystem oder Ihrer Fehlerdatenbank. Im Moment beträgt die Anzahl der Fehler Null.

Zusammenfassung der studierten Metriken

Durch die Implementierung und regelmäßige Überwachung dieser Metriken können wir die Qualität unseres Projekts kontinuierlich verbessern und sicherstellen, dass wir ein stabiles, fehlerfreies und benutzerfreundliches Produkt liefern.