

Zusammenfassung des Quellcode-Refactorings – Clean-Code-Praxis

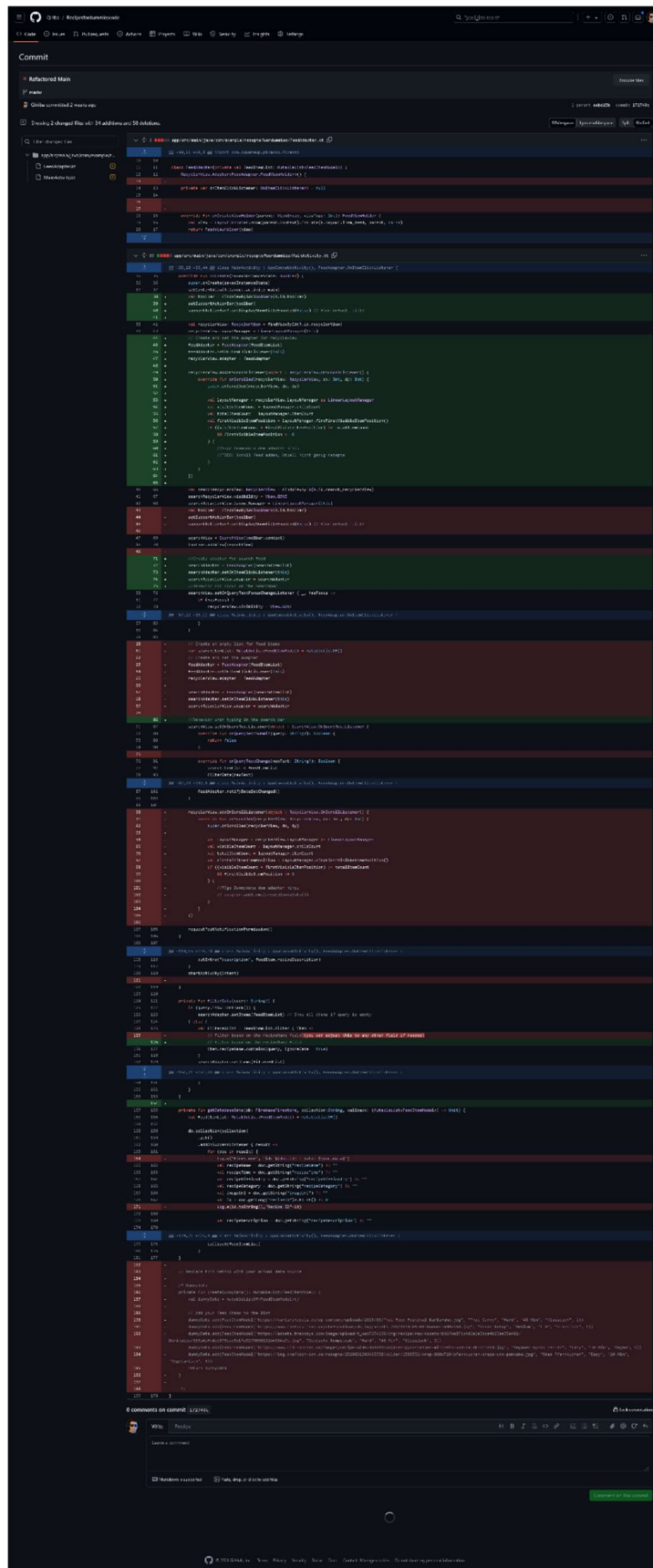
Einheitliche Struktur: Die konsistente Benennung der Variablen stellt sicher, dass der Code eine einheitliche Struktur aufweist, was wiederum die Wartbarkeit verbessert.

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>

Qiriba / Recipesfordummiescode

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Commit

Refactored Main

master

Qiriba committed 2 weeks ago

1 parent 172749c · commit ff544d3

Showing 1 changed file with 22 additions and 1 deletion.

Whitespace Ignore whitespace Split Unfold

app/src/main/java/com/example/recipesfordummies/MainActivity.kt

```
... @@ -1,12 +1,15 @@
1 package com.example.recipesfordummies
2
3 import android.Manifest
4 + import android.content.Context
5 import android.content.Intent
6 import android.content.pm.PackageManager
7 import android.os.Build
8 import android.os.Bundle
9 import android.util.Log
10 import android.view.View
11 + import android.view.inputmethod.EditorInfo
12 + import android.view.inputmethod.InputMethodManager
13 import androidx.appcompat.app.AppCompatActivity
14 import androidx.appcompat.widget.SearchView
15 import androidx.appcompat.widget.Toolbar
16
17 @@ -19,7 +22,6 @@ import com.google.firebase.ktx.Firebase
18 import com.google.firebase.firestore.FirebaseFirestore
19
20
21
22 -
23 class MainActivity : AppCompatActivity(), FeedAdapter.OnItemClickListener {
24
25     companion object {
26
27         @@ -72,6 +74,26 @@ class MainActivity : AppCompatActivity(), FeedAdapter.OnItemClickListener {
28         searchAdapter = FeedAdapter(searchItemList)
29         searchAdapter.setOnItemClickListener(this)
30         searchRecyclerView.adapter = searchAdapter
31         val searchText = findViewById<TextView>(R.id.search_text)
32         searchText.setOnClickListener { v, actionId, event ->
33             if (actionId == EditorInfo.IME_ACTION_SEARCH) {
34                 filterData(searchText.text.toString())
35                 hideKeyboard(searchEditText)
36                 true
37             } else {
38                 false
39             }
40         }
41         searchView.setOnClickListener {
42             filterData(null) // Reset to full list when search is closed
43             false
44         }
45
46         //Behavior for focus on the searchbar
47         searchView.setOnQueryTextFocusChangeListener { _, hasFocus ->
48             if (hasFocus) {
49
50                 @@ -88,6 +96,7 @@ class MainActivity : AppCompatActivity(), FeedAdapter.OnItemClickListener {
51             } else {
52                 searchRecyclerView.visibility = View.GONE
53                 recyclerView.visibility = View.VISIBLE
54                 searchView.onActionViewCollapsed()
55             }
56         }
57
58         @@ -175,4 +182,8 @@ class MainActivity : AppCompatActivity(), FeedAdapter.OnItemClickListener {
59             callback(feedItemList)
60         }
61     }
62
63     fun Context.hideKeyboard(view: View) {
64         val im = getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
65         im.hideSoftInputFromWindow(view.windowToken, 0)
66     }
67 }
```

0 comments on commit ff544d3

Write Preview

Leave a comment

Markdown is supported Paste, drop, or click to add files

Comment on this commit

© 2024 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information

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 standardmäßig bereits erstellt worden.

Hier ist ein Implementierter Test in unserem Code

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

11 11  @ -11,7 -11,22 @@ import org.junit.Assert.*
12 12  */
13 13  class ExampleUnitTest {
14 14  @Test
15 15  fun addition_isCorrect() {
16 16      assertEquals(4, 2 + 2)
17 17  }
18 18  fun getItemId() {
19 19      val feedItemList: MutableList<FeedItemModel> = createDummyData()
20 20      val adapter = FeedAdapter(feedItemList)
21 21      assertEquals(adapter.getItemId(1), -693801673)
22 22      assertEquals(adapter.getItemId(0), 1117640830)
23 23  }
24 24  fun createDummyData(): MutableList<FeedItemModel> {
25 25      val dummyData = mutableListOf<FeedItemModel>()
26 26
27 27      // Add your feed items to the list
28 28      dummyData.add(FeedItemModel("https://karlsruhepuls.de/wp-content/uploads/2023/06/Thai-Food-Festival-Karlsruhe.jpg", "Thai Curry", "Hard", "45 Min", "Klassisch", 1))
29 29      dummyData.add(FeedItemModel("https://www.chefsculinar.de/chefsculinar/ds_img/assets/700/2014-09-04-Doener-690x468.jpg", "Döner Kebap", "Medium", "1 H", "Klassisch", 2))
30 30      dummyData.add(FeedItemModel("https://assets.tmcocys.com/image/upload/t_web/767x639/img/recipe/ras/Assets/b36f8e87cd4d6e63dce4b23aa35e481/Derivates/563a4efc4ab575cad5db7a9279896132b4334a7c.jpg",
31 31      "Deutsche Rumpsteak", "Hard", "45 Min", "Klassisch", 3))
32 32      dummyData.add(FeedItemModel("https://www.lidl-kochen.de/images/recipe-wide/868844/veganer-gyrosteller-mit-reis-und-salat-311084.jpg", "Veganer Gyros Teller", "Easy", "30 Min", "Vegan", 4))
33 33      dummyData.add(FeedItemModel("https://img.chefkoch-cdn.de/rezepte/2529831396465558/bilder/1589532/crop-968x720/pfannkuchen-crepe-und-pancake.jpg", "Omas Pfannkuchen", "Easy", "20 Min", "Vegetarisch", 5))
34 34      return dummyData
35 35  }
36 36  }
```

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.