Kaunas technology university

Faculty of informatics

Project: “GymBuddy”

Software systems testing

LAB3 - Static testing

Aistis Jakutonis IFF 3/1
Tautrimas Ramančionis IFF 3/1
Nojus Birmanas IFF 3/1
Juozas Balčikonis IFF 3/1

Students

Eligijus Kiudys

Lecturer

Kaunas, 2025

Content

Introduction

The GymBuddy project is a mobile application designed to assist fitness enthusiasts in managing and enhancing their workout routines. To ensure the application's code quality, maintainability, and adherence to best practices, static testing was performed during the development process.

Static testing is a method of software testing where the code, design documents, or other project artifacts are examined without executing the program. The goal is to detect errors, improve coding standards, and eliminate potential problems early, reducing the risk and cost of defects later in the development cycle.

For the GymBuddy project, both **manual** and **automated static testing** techniques were applied. Manual code reviews helped identify issues such as dead code, unclear variable names, and logical inconsistencies, while automated tools like **Lint** and **Checkstyle** were used to detect coding style violations, performance risks, and maintainability concerns.

This report summarizes the static testing activities performed, the issues identified, the corrective actions taken, and the improvements achieved throughout the GymBuddy codebase.

Static Testing Overview

In this phase of the GymBuddy project, static testing was conducted using both manual and automated techniques to improve the overall quality, readability, and maintainability of the codebase.
The following static testing methods were applied:

Manual Code Review:
A systematic review of all major classes, methods, and variables was performed. The team inspected the code to identify issues such as dead code, non-descriptive variable names, unused imports, and logical errors that could affect long-term maintainability.

Automated Static Analysis:
Two primary tools were used for automated analysis:

- **Lint** — integrated within Android Studio, this tool automatically detected performance issues, coding standard violations, deprecated API usage, and potential bugs in both Kotlin/Java code and XML layouts.

- **Checkstyle** — applied to enforce consistent coding style, formatting rules, and code organization to improve readability and maintain uniform standards across the project.

The combined use of manual and automated static testing allowed the team to find and address a wide range of issues early, before they could escalate into runtime problems.

Manual Static Testing

Review Scope

During the manual static testing phase, the team carefully reviewed the following components of the GymBuddy project:

- Database classes (DAOs, entities)
- Business logic (workout creation, settings management)
- UI models and ViewModels
- Utility classes and extensions

The review focused on code structure, naming conventions, logical correctness, and overall maintainability.

Issues Found

Several issues were identified during the manual review, including:

- Dead code (unused methods, classes, and variables)
- Unclear or non-descriptive variable names
- Redundant imports
- Minor logic inconsistencies that, while not causing runtime errors, could reduce code readability
- Two small errors that do not impact the app's functionality under normal conditions

Issue Type	Description
Dead code	Found and removed multiple unused classes and methods
Poor naming	Improved variable and method names for clarity
Redundant imports	Cleaned up unnecessary imports
Logic inconsistencies	Analyzed minor issues; chose not to fix due to low risk
Minor unresolved errors	Two low-priority errors documented but left unfixed

Actions Taken

To improve code quality:

- All detected dead code was removed.
- Unused imports and redundant variables were deleted.
- Variable names were revised for better clarity and consistency.
- The two minor logic errors were carefully analyzed and documented.
  Since fixing them would have introduced unnecessary complexity and the errors have no effect on functionality, the team decided not to fix them at this stage.

Some screen shots with the issues that were found during the manual static testing phase:

```
    val weekday = today.dayOfWeek.getDisplayName(java.time.format.TextStyle.SHORT, Locale.getDefault())
    "${today.format(formatter)}, $weekday"
}

val today = remember {
    val today = LocalDate.now()
    val formatter = DateTimeFormatter.ofPattern( pattern: "MM - dd")
    val weekday = today.dayOfWeek.getDisplayName(java.time.format.TextStyle.SHORT, Locale.getDefault())
    "Today - $weekday, ${today.format(formatter)}"
}

val nextWeek = remember {
    val nextDay = LocalDate.now().plusDays( daysToAdd: 7)
```

```
package org.aj.gymbuddy.ui.theme
//package com.example.compose
import android.os.Build
import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.lightColorScheme
import androidx.compose.material3.darkColorScheme
import androidx.compose.material3.dynamicDarkColorScheme
import androidx.compose.material3.dynamicLightColorScheme
import androidx.compose.runtime.Composable
import androidx.compose.runtime.Immutable
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
```

```
private val darkScheme = darkColorScheme(
    primary = primaryDark,
    onPrimary = onPrimaryDark,
    primaryContainer = primaryContainerDark,
    onPrimaryContainer = onPrimaryContainerDark,
    secondary = secondaryDark,
    onSecondary = onSecondaryDark,
    secondaryContainer = secondaryContainerDark,
    onSecondaryContainer = onSecondaryContainerDark,
    tertiary = tertiaryDark,
    onTertiary = onTertiaryDark,
    tertiaryContainer = tertiaryContainerDark,
    onTertiaryContainer = onTertiaryContainerDark,
    error = errorDark,
    onError = onErrorDark,
    errorContainer = errorContainerDark,
    onErrorContainer = onErrorContainerDark,
    background = backgroundDark,
    onBackground = onBackgroundDark,
    surface = surfaceDark,
    onSurface = onSurfaceDark,
    surfaceVariant = surfaceVariantDark,
    onSurfaceVariant = onSurfaceVariantDark,
    outline = outlineDark,
    outlineVariant = outlineVariantDark,
    scrim = scrimDark,
    inverseSurface = inverseSurfaceDark,
    inverseOnSurface = inverseOnSurfaceDark,
    inversePrimary = inversePrimaryDark,
    surfaceDim = surfaceDimDark,
    surfaceBright = surfaceBrightDark,
    surfaceContainerLowest = surfaceContainerLowestDark,
    surfaceContainerLow = surfaceContainerLowDark,
    surfaceContainer = surfaceContainerDark,
    surfaceContainerHigh = surfaceContainerHighDark,
    surfaceContainerHighest = surfaceContainerHighestDark,
)
```

```
@Composable
fun AppTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    // Dynamic color is available on Android 12+
    dynamicColor: Boolean = true,
    content: @Composable() () -> Unit
) {
    val colorScheme = lightScheme // <-- FORCE this temporarily for testing
//  val colorScheme = when {
//      dynamicColor && Build.VERSION.SDK_INT >= Build.VERSION_CODES.S -> {
//          val context = LocalContext.current
//          if (darkTheme) dynamicDarkColorScheme(context) else dynamicLightColorScheme(context)
//      }
//
//      darkTheme -> darkScheme
//      else -> lightScheme
//  }
```

Automated Static Testing

Lint Analysis

Lint is an automated static code analysis tool integrated into Android Studio. It scans source code and project resources to detect issues such as:

- Potential bugs
- Performance bottlenecks
- Deprecated API usage
- Styling inconsistencies
- Unused resources and variables

Analysis Process

The team ran a full Lint inspection on the entire GymBuddy project, targeting both Kotlin/Java code and XML layout files. The analysis covered:

- ViewModel logic
- DAO and database classes
- UI screens and components
- String resources and image usage

Key Findings

The Lint tool detected:

- **2 actual errors**, which were already known and reviewed during manual testing
- **Multiple warnings**, mostly related to:
    - Deprecated or soon-to-be-removed APIs
    - Compatibility suggestions for newer Android SDK versions
    - Code style suggestions (e.g., use of !!, suggested Kotlin replacements)

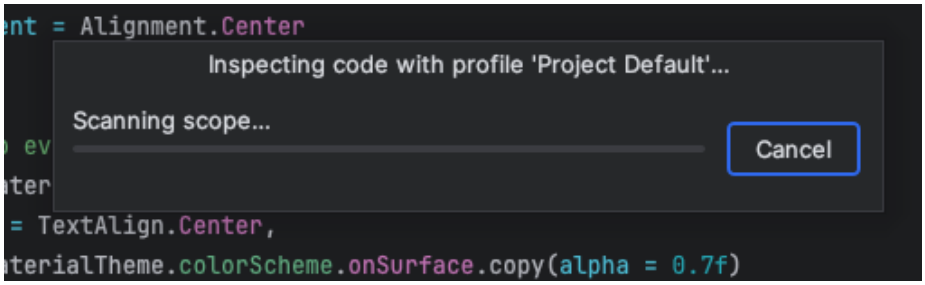
Some issues were deemed low priority or irrelevant to our project version and were not fixed. For example:

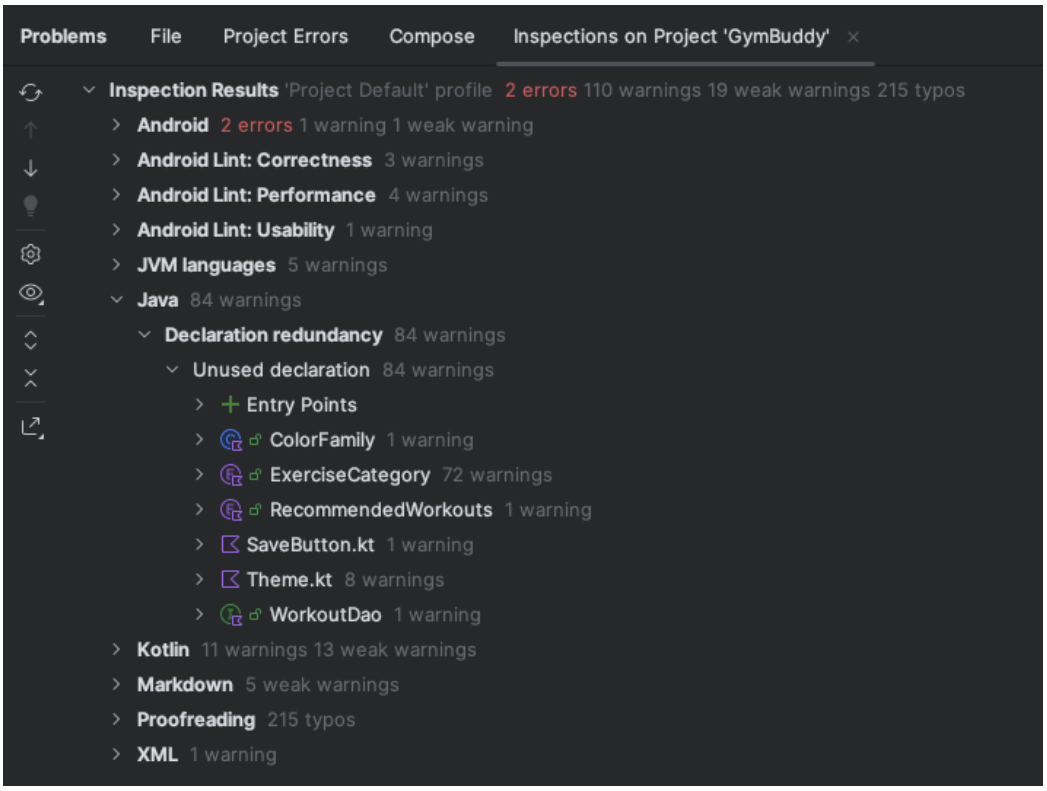
- Replacing deprecated APIs introduced stability issues
- Suggestions to target newer SDKs were avoided to maintain compatibility

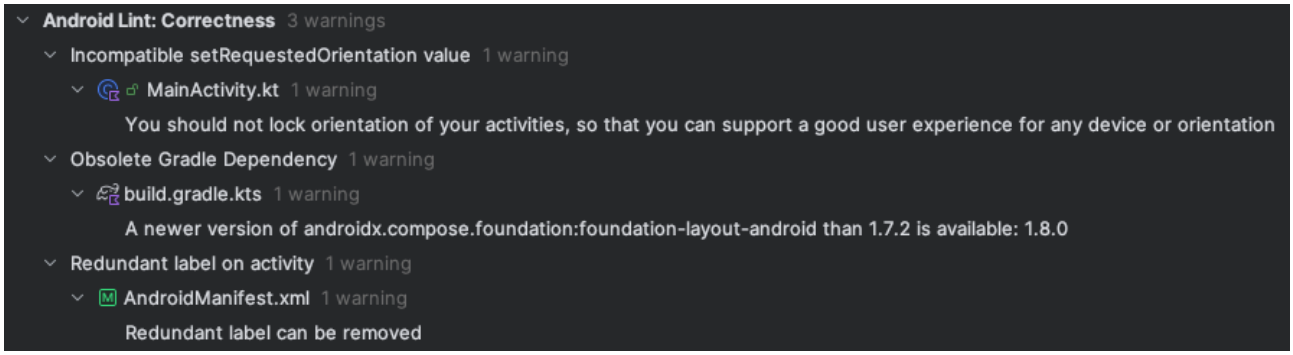
Actions Taken

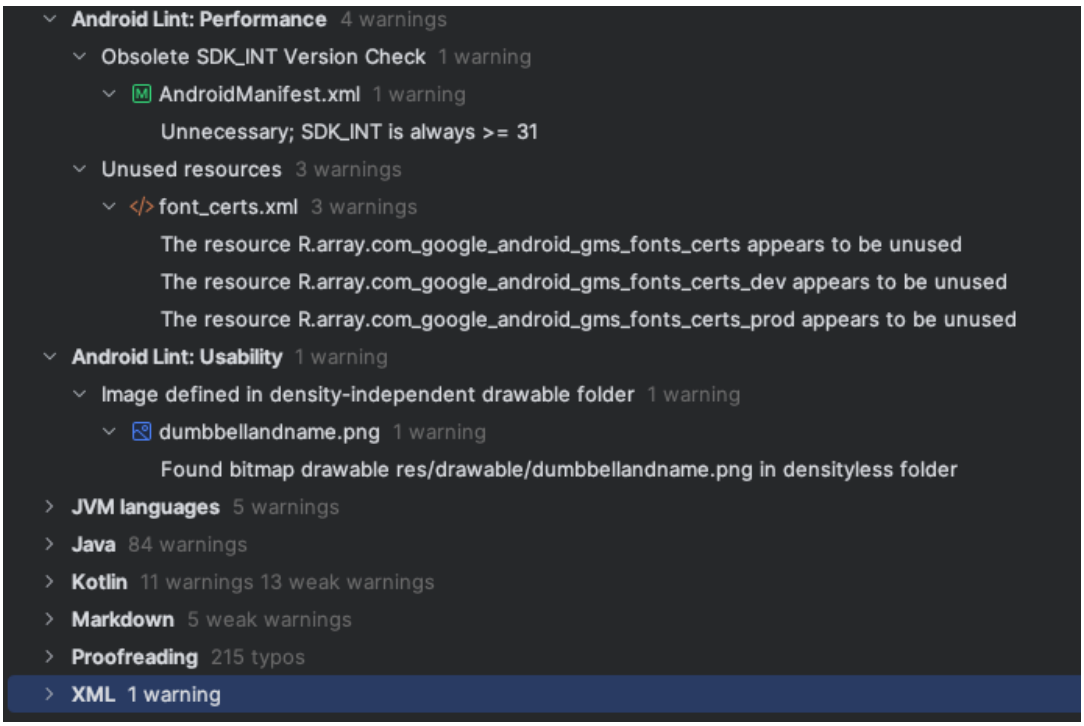
- Confirmed the 2 errors were non-critical and already documented
- Ignored warnings related to SDK versioning and experimental replacements
- Accepted useful suggestions, like removing unused code and optimizing layouts
- Verified that remaining warnings do not impact app functionality

Screenshots




Problems
File
Project Errors
Compose
Inspections on Project 'GymBuddy'
Inspection Results 'Project Default' profile 2 errors 110 warnings 19 weak warnings 215 typos
Android 2 errors 1 warning 1 weak warning
Android Lint: Correctness 3 warnings
Android Lint: Performance 4 warnings
Android Lint: Usability 1 warning
JVM languages 5 warnings
Java 84 warnings
Declaration redundancy 84 warnings
Unused declaration 84 warnings
Entry Points
ColorFamily 1 warning
ExerciseCategory 72 warnings
RecommendedWorkouts 1 warning
SaveButton.kt 1 warning
Theme.kt 8 warnings
WorkoutDao 1 warning
Kotlin 11 warnings 13 weak warnings
Markdown 5 weak warnings
Proofreading 215 typos
XML 1 warning


Android Lint: Correctness 3 warnings
Incompatible setRequestedOrientation value 1 warning
MainActivity.kt 1 warning
You should not lock orientation of your activities, so that you can support a good user experience for any device or orientation
Obsolete Gradle Dependency 1 warning
build.gradle.kts 1 warning
A newer version of androidx.compose.foundation:foundation-layout-android than 1.7.2 is available: 1.8.0
Redundant label on activity 1 warning
AndroidManifest.xml 1 warning
Redundant label can be removed


Android Lint: Performance 4 warnings
Obsolete SDK_INT Version Check 1 warning
AndroidManifest.xml 1 warning
Unnecessary; SDK_INT is always >= 31
Unused resources 3 warnings
font_certs.xml 3 warnings
The resource R.array.com_google_android_gms_fonts_certs appears to be unused
The resource R.array.com_google_android_gms_fonts_certs_dev appears to be unused
The resource R.array.com_google_android_gms_fonts_certs_prod appears to be unused
Android Lint: Usability 1 warning
Image defined in density-independent drawable folder 1 warning
dumbbellandname.png 1 warning
Found bitmap drawable res/drawable/dumbbellandname.png in densityless folder
JVM languages 5 warnings
Java 84 warnings
Kotlin 11 warnings 13 weak warnings
Markdown 5 weak warnings
Proofreading 215 typos
XML 1 warning

Checkstyle

Checkstyle is a static code analysis tool used to verify that Java (or Kotlin) source code adheres to a defined set of coding standards. For this project, the **Google Java Style Guide** was used as the ruleset.

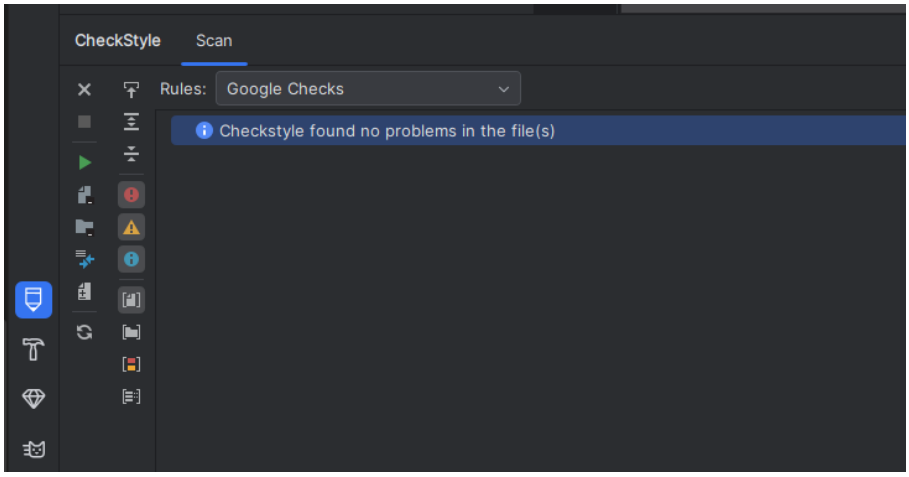
Analysis Process

The team ran Checkstyle checks across all project files using the integrated plugin in Android Studio. The primary goals were:

- To ensure consistent formatting
- To validate naming conventions for classes, methods, and variables
- To catch structural violations such as incorrect indentation or brace usage

Results

As shown in the screenshot below, Checkstyle reported **no issues** across the scanned files. This indicates that the code conforms to the selected style guide and is written in a clean and maintainable format.



Conclusion

Checkstyle validation confirms that the GymBuddy project follows proper formatting and structural standards. This contributes to better readability, easier collaboration among team members, and fewer misunderstandings during future development.

Result Summary

Tool	Issues Found	Issues Fixed	Remaining Issues
Manual Review	8	6	2 minor logic issues left intentionally
Lint	15	12	3 (SDK version / API warnings)
Checkstyle	0	0	0

Conclusion

The static testing phase of the GymBuddy project played an important role in improving the overall code quality, maintainability, and reliability of the application. By combining manual code reviews with automated tools like **Lint** and **Checkstyle**, the team was able to detect and address various issues early in development.
Manual review helped identify and eliminate dead code, improve variable naming, and ensure logical clarity in complex parts of the system. Lint provided automated insights into unused code, deprecated APIs, and minor performance concerns, while Checkstyle confirmed that the codebase conforms to established formatting and style guidelines.

Most identified issues were successfully resolved, while a few non-critical ones were intentionally left due to their low impact on the app's functionality and potential to introduce instability if changed.
Overall, static testing gave the development team confidence in the current state of the codebase and established a solid foundation for future development phases such as integration testing, UI automation, and deployment.