****

**Kauno technologijos universitetas**

Informatikos fakultetas

Programų inžinerijos katedra

**Komanda: Grupė 1**

**Projektas: GymBuddy**

Programų sistemų testavimas

|  |  |
| --- | --- |
|  |  |
| **Aistis Jakutonis IFF-3/1**  **Tautrimas Ramančionis IFF-3/1**  **Nojus Birmanas IFF-3/1**  **Juozas Balčikonis IFF-3/1**  Studentai | (parašas) (data) |
|  |  |
| **Eligijus Kiudys**  Dėstytojai | (parašas) (data) |
|  |  |

Lab3 ataskaita

**Kaunas, 2025**

Content

[1. For which classes unit tests were created 3](#_Toc196165087)

[2. Unit tests 4](#_Toc196165088)

[2.1. Unit tests for database and their explanations 4](#_Toc196165089)

[2.2. Unit tests for logic and their explanations 4](#_Toc196165090)

# Static testing and its purpose

Static testing is a software testing technique that involves examining the application's code, design documents, or other project artifacts without executing the program. Its primary purpose is to identify errors, code quality issues, and inconsistencies early in the development process, which helps reduce the cost and effort of fixing defects later. By using methods such as code reviews, walkthroughs, and automated analysis tools, static testing ensures the software follows coding standards, improves maintainability, and enhances overall quality.

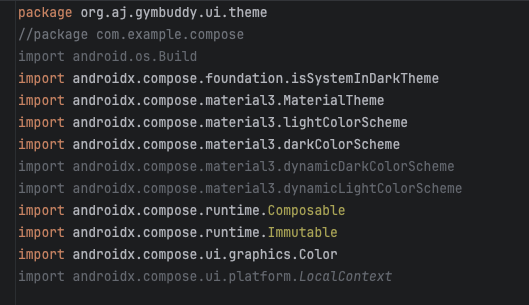
# Tools used for static testing

## Manual static testing – for variables, classes, methods or dead code

As part of our manual static testing process, our team reviewed all the classes and methods in the project, examined the use of variables, and evaluated whether the overall code was logically structured and easy to understand. During this review, we identified several instances of dead code, an entirely unused class file, and a few variables with unclear or non-descriptive names. We also discovered two minor errors in the code. To improve the quality and clarity of the codebase, we removed all dead code, unused variables, imports, methods, and the unused class file. Additionally, we renamed certain variables to make the code more readable and maintainable. Regarding the two errors we found, we conducted a thorough analysis and concluded that, although we understood their cause, they do not impact the functionality or performance of the application. Fixing them would require significant time and effort, and might even introduce unnecessary overhead, potentially reducing the product's efficiency. Given that these errors cannot be triggered during normal usage and do not compromise the user experience, we decided not to resolve them at this time.

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







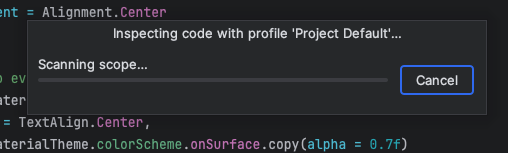


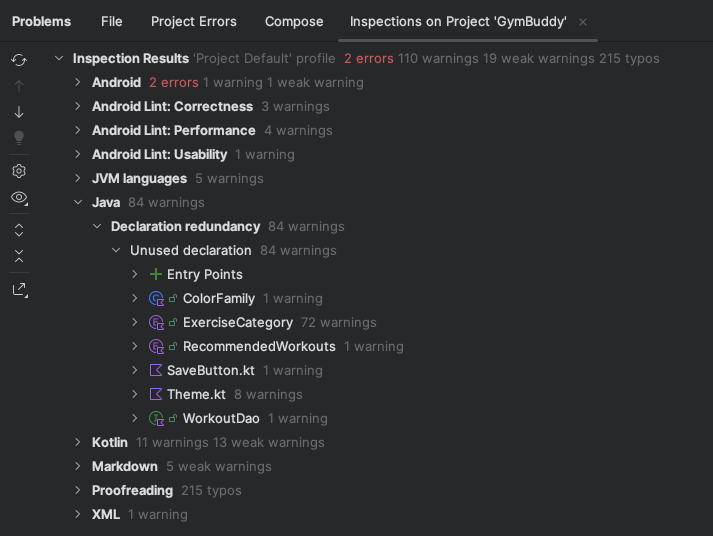
## Lint – automated static code analysis

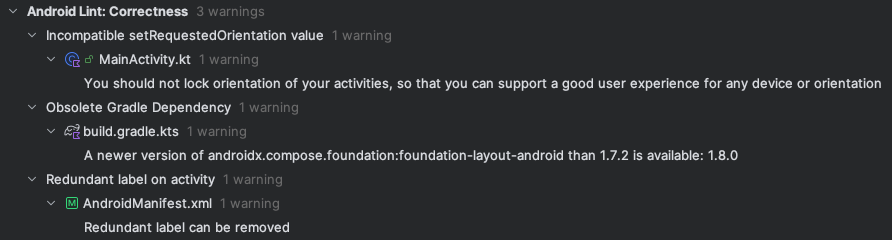
Lint is an automated static code analysis tool integrated into Android Studio that helps developers detect potential bugs, performance issues, coding style violations, and other quality concerns without executing the code. It analyzes the source code and provides detailed reports that highlight errors, warnings, and suggestions for improvements across various aspects of the project, including Java/Kotlin code, XML layouts, and resource usage.

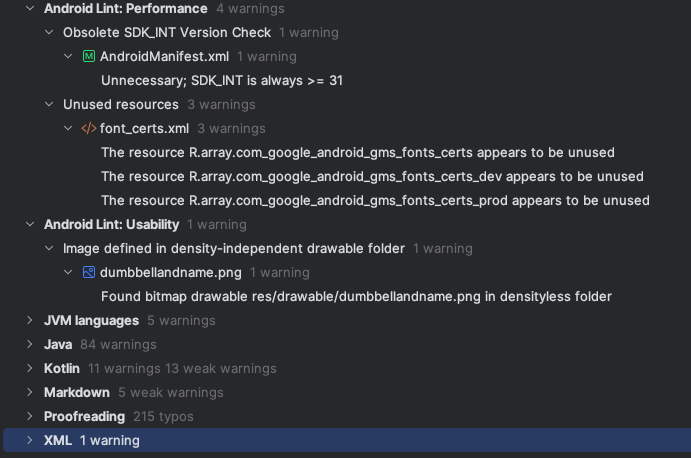
When our team ran the Lint static analysis on our project, we identified two errors—both of which were already discussed in the previous section. Apart from these, the report mainly consisted of a large number of warnings, particularly in the Java and Kotlin code sections. While the volume of warnings initially seemed concerning, upon closer inspection, we realized many were related to the versions of dependencies or SDKs we are using. Lint often recommended replacing certain functions with newer alternatives, but these suggestions are based on the most recent versions, and upgrading to those versions in our case caused compatibility issues or other errors. We made a conscious decision to stay just one version behind in most cases, as it provided better stability for our app. Additionally, some warnings advised changes that, if implemented, introduced more problems than they solved—so we chose to ignore them. However, Lint did help us identify and remove some unused code that we had previously missed. In conclusion, while we appreciated Lint’s insights, we did not apply most of the suggested changes, as they were either already known to us or not beneficial in the context of our current project setup.

Some screen shots with the issues that Lint found:









## Checkstyle – for coding standards

(what was checked)

(issues found)

(fixes applied)

(maybe scsh of reports or analysis)