Kaunas technology university

Faculty of informatics

Project: “GymBuddy”

Test plan

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 3
Tests scope 3
Test strategies 3
Prerequisites 4
Test priorities 5
Test goals 6
Test techniques 7
Tests management 9
Results 9
Testing environment 10
Test scripts 11
    Test script 1: Creating a Workout and Adding Activities 12
    Test script 2: Adding a Workout to the Calendar 13
Testing schedule 14

Introduction

This document outlines the test plan for **“GymBuddy”,** a mobile app for Android. The goal is to ensure the app meets functional requirements, identify issues, and verify usability.

“GymBuddy” is a designed to help users plan, track, and optimize their workouts. It allows users to create custom workout routines, schedule sessions in a built-in calendar. The app includes guided exercises, progress tracking, and integration with fitness-related features to enhance the overall gym experience. It is being developed using **Android Studio and Kotlin** for native Android compatibility. Development focuses on delivering a reliable, offline-first experience on Android devices.

Tests scope

Acceptance testing consists of:
“GymBuddy” mobile application (Android), version 1.0.
Use case models for users (gym-goers, fitness enthusiasts). The goal of testing is to validate functionality, usability, security, and performance. Users' varied working habits will be disregarded during testing.

Test strategies

Acceptance testing will verify system functionality, stability and performance. Other testing phases will be completed before acceptance testing. Testing will focus on user experience to ensure “GymBuddy” meets quality expectations. Additional testing includes:

- Unit Testing - Testing individual features.
- Static Testing – Reviewing code, design, and documentation to identify defects early.
- Performance Testing - Assessing speed and stability.
- Usability Testing - Checking ease of use.

Prerequisites

Before testing begins, the following prerequisites must be in place to ensure a smooth and efficient testing process:

1. **Fully Functional App Prototype**
   – The latest working version of the "GymBuddy" Android application must be available for testing.
   – Core features, including workout creation, scheduling, and settings management, should be implemented and accessible.
2. **Defined Test Cases**
   – A complete set of structured test cases should cover all major functionalities of the app.
   – Each test case must specify expected inputs, steps to perform, and expected outcomes.
3. **Testing Environment Setup**
   – **Devices**: Access to at least one physical Android device and Android Studio emulators.
   – **OS Versions**: Ensure compatibility with Android 10 and above.
   – **Network Conditions**: A stable internet connection for testing any cloud-based or sync-related features, if applicable.
4. **Bug Tracking System**
   – Use a centralized method (GitHub and Jira) to record bugs.
   – Each reported issue should include reproduction steps, screenshots (if needed), and assigned priority.
5. **Dedicated Testers**
   – Assign at least 2–3 team members to testing tasks.
   – Testers should execute predefined test cases and document all findings clearly and consistently.
6. **Issue Resolution Plan**
   – Developers must be available to address bugs as they are reported.
   – Establish a workflow for retesting and verifying resolved issues before closing them.
7. **User Feedback Collection**
   – Invite a few external users (e.g., friends or classmates) to perform usability testing on Android devices.
   – Gather feedback on the app's functionality, visual design, and overall user experience.

By ensuring these prerequisites are met, we can conduct structured, thorough, and effective testing of the "GymBuddy" Android application.

Test priorities

Testing will focus on the most critical aspects to ensure that "GymBuddy" operates smoothly, delivers a high-quality user experience, and performs reliably on Android devices. The following priorities guide the testing approach:

1. Core Functionalities (High Priority)
    - Ensure that all planned features work as expected.
    - Validate key app functionalities, including:
        - Workout Session Scheduling – Users should be able to add, edit, and delete sessions.
        - Workout planning – Check if the user is able to create, edit and delete his own workouts.
    - Verify that inputs are correctly processed and stored.

2. Usability and User Experience (High Priority)
    - Ensure the app has an intuitive and easy-to-navigate interface.
    - Test button placements, font sizes, and color schemes for readability.
    - Evaluate the efficiency of workflows (e.g., how easily users can add a workout).
    - Gather feedback from test users to identify confusing elements.

3. Performance and Stability (High Priority)
    - Test the app on different devices to ensure smooth performance.
    - Measure response times for key actions (e.g., adding a workout session).
    - Identify any crashes or slowdowns during typical use.

4. Compatibility and Responsiveness (Medium Priority)
    - Test "GymBuddy" on different screen sizes and resolutions.
    - Ensure UI elements are correctly displayed across devices.

5. Security and Data Integrity (Medium Priority)
    - Verify that user data (workout logs, preferences) is stored and retrieved correctly.
    - Test how the app handles incorrect or incomplete input data (e.g., leaving fields blank).
    - Ensure that saved workout data persists after app restarts.

6. Edge Case and Stress Testing (Low Priority but Useful)
    - Check how the app handles extreme input values.
    - Simulate high usage (e.g., rapidly adding and deleting multiple workouts).
    - Assess how the app behaves with low storage or poor network conditions (if syncing is involved).

Test goals

The primary goal of testing “GymBuddy” is to ensure that the app functions correctly, delivers a smooth and intuitive user experience, and performs reliably across various devices. The testing process is structured to identify and resolve potential issues before release, ensuring a high-quality product. The following specific goals guide our testing approach:

1. Functional Validation
    - Ensure all core features (workout scheduling, planning) work as expected.
    - Verify that users can easily add, edit, and delete workout sessions.
    - Test various user inputs, including unexpected or invalid data, to ensure the system handles them gracefully without crashes or data corruption.
2. Integration and compatibility testing
    - Validate that different components of the app, such as backend services, databases, and external integrations, work together seamlessly.
    - Test data flow and synchronization between different modules to ensure consistency and prevent data loss or corruption.
    - Confirm compatibility across various platforms and different screen sizes, ensuring a uniform experience for all users.
    - Check for smooth integration with wearable devices, fitness trackers, and cloud services, if applicable.
3. Performance and Stability
    - Validate app responsiveness, load times across various devices and operating systems.
    - Identify and resolve potential crashes, freezes, slow responses, or memory leaks that could impact user experience.
    - Ensure the app maintains stable performance under varying levels of usage, including high user traffic and extended usage sessions.
4. Usability Testing
    - Evaluate the app’s interface for ease of use and navigation.
    - Gather feedback from test users to identify areas for improvement.
    - Ensure accessibility for different user demographics (e.g., color contrast for visibility).
5. Static Testing
    - Code Reviews – Conducting peer reviews to identify syntax errors, security vulnerabilities, and coding standard violations.
    - Design Reviews – Assessing UI/UX wireframes, database schemas, and system architecture for consistency and efficiency.
    - Static Code Analysis – Using automated tools to check for security flaws, performance bottlenecks, and maintainability issues.

Test techniques

To ensure the mobile application functions reliably and meets quality standards, a combination of testing methods will be applied. These methods are designed to validate individual components, overall system behavior, and user experience. The primary testing categories include:

1. **Unit testing**

   Unit testing focuses on verifying the behavior of individual components such as functions, methods, or classes. Each unit will have dedicated test cases to confirm it performs as expected. This allows developers to quickly identify and fix bugs at the code level, ensuring core functionality works correctly in isolation.

2. **Integration testing**

   Integration tests assess how different modules or features interact with each other. This helps detect issues that may arise when components are combined, even if they passed their individual unit tests. The goal is to ensure seamless collaboration between different parts of the application.

3. **Static analysis**

   Static analysis involves reviewing the application's source code without executing it. This technique helps identify potential errors, bad practices, and maintainability concerns early in the development process—before the application is even run.

4. **Performance testing**

To evaluate the app's responsiveness and efficiency, several performance-related tests will be conducted:

Response Time – Measures how quickly the app reacts to user input.
Resource Usage – Analyzes how efficiently the app uses device resources such as CPU, memory, and battery.
Stability Testing – Assesses whether the app can run for prolonged periods without crashing or freezing.
Stress Testing – Simulates high-load scenarios, such as managing large datasets or heavy user interaction.

5. **Usability testing**

Usability testing focuses on the overall user experience. It ensures the app is intuitive, easy to navigate, and accessible. Feedback will be collected from real users or testers to identify friction points, and the UI/UX will be adjusted as needed to improve satisfaction.

6. Testing Techniques

To streamline the testing process, the following methods will be used:

- **Scripted Test Cases** – Predefined scenarios with known input and expected output to verify consistency and correctness.
- **Exploratory Testing** – Testers will freely interact with the app using unscripted inputs to uncover unexpected bugs and behavior.
- **Usability Evaluation** – A checklist-based assessment of the app's interface, accessibility, and overall user-friendliness.
- **Performance Monitoring** – Gathering and analyzing data related to responsiveness, stability, and resource consumption to ensure the app performs efficiently under typical and edge-case scenarios.

Tests management

In order to better manage the testing, we decided to split our team into a few roles:

1. Quality assurance lead - oversees the entire testing process, ensures testing strategies align with project goals, assigns tasks to testers and maintains high quality standards.
2. Main testers - conduct hands-on testing, identify and document bugs, validate fixes, and report findings to the QA lead.
3. Product manager - defines product requirements, ensures testing aligns with business objectives, prioritizes fixes, and facilitates communication between team members.

Results

Expected testing results

The testing process aims to determine whether the software meets the defined functional and non-functional requirements. The key objectives include:

- Verifying that all intended functions operate according to the specifications.
- Ensuring the system meets security, performance, and reliability criteria.
- Evaluating system stability under real-world usage conditions.
- Confirming that the software is ready for deployment or further updates.

The following outputs should be available after the completion of testing:

- The test plan document, including all modifications made throughout the testing process.
- Change requests – a record of software modifications resulting from updated requirements or identified defects during testing. All changes will be documented in JIRA.

Testing environment

Testing Tools and Frameworks

The following tools will be used to support the development and testing process:

Android Studio (latest stable version)

The main IDE used for building, testing, and debugging the application. It includes integrated tools for profiling, emulation, and test automation.

JUnit *(built-in with Android Studio)*

A widely used framework for unit testing in Android. It enables verification of individual components such as functions, classes, and view models.

Lint and Static Code Analysis *(via Android Studio / Gradle plugins)*

Built-in tools used to analyse source code for potential bugs, styling issues, and performance concerns before execution.

Android Profiler

A performance monitoring tool included in Android Studio to track CPU, memory, battery usage, and network activity during app runtime.

First testing workstation
CPU: Apple M1 Max 3,6 GHz
RAM: 32 GB
Storage: 1 TB SSD
Operating system: macOS Sequoia 15.3.1

Second testing workstation
CPU: Intel Core i7 4790 3,6 GHz
RAM: 16 GB
Storage: 1TB HDD + 225 GB SSD
Operating system: Windows 10 Home

Third testing workstation
CPU: AMD Ryzen 5 5600X 3,7 GHz / 4,6 GHz
RAM: 32 GB
Storage: 1 TB SSD
Operating system: Windows 11 Professional

Fourth testing workstation
CPU: AMD Ryzen 7 3700X 4,4 GHz
RAM: 32 GB
Storage: 500 GB SSD + 2 TB HDD
Operating system: Windows 11 Home

Fifth testing workstation
CPU: Intel Core i7 6600U 2,6 GHz / 3,4 GHz
RAM: 16 GB
Storage: 250 GB SSD
Operating system: Windows 11 Professional

All workstations must have all the software listed at the top.

Test scripts

Introduction

The following test scripts are designed to verify the core functionalities of the “GymBuddy” mobile application. These scripts correspond to various use case scenarios but are not detailed within this document; instead, they are linked to the relevant system documentation. Each test script follows a structured approach to ensure consistency in testing and reliable results.

Each test script consists of:
Description – A brief overview of the test scenario.
Initial Data – The state of the application before executing the test.
Test Steps – A sequence of actions that the tester must perform.
Test Cases – Input data and expected results for validation.

Test script 1: Creating a Workout and Adding Activities

Description: This script tests whether a user can create a workout and add exercises to it.

Initial Data:

- The “GymBuddy” app is installed and opened.
- The user is logged into their account.
- The "Workouts" section is accessible.

Test Steps:

1. Navigate to the **"Workouts"** section.
2. Click on **"Create New Workout"**.
3. Enter a **workout name** (e.g., "Upper Body Strength").
4. Click **"Add Activities"**.
5. Select exercises from the predefined list (e.g., Bench Press, Pull-ups) or create a custom exercise.
6. Define **sets, repetitions, and weight** for each exercise.
7. Click **"Save"**.
8. Verify that the workout is saved and appears in the workout list.

Test Cases:

Input Data	Expected Result	Comments
Workout Name: "Full Body Strength"	Workout is created successfully	Workout appears in the list
Added exercises: Squats, Deadlifts, Push-ups	Exercises are listed correctly within the workout	Each exercise shows sets, reps, and weight
Custom Exercise: "Kettlebell Swings"	Exercise is saved correctly	Shows in custom exercises

Test script 2: Adding a Workout to the Calendar

Description: This script ensures that a user can schedule a workout in the calendar.

Initial Data:

- The user is logged into their “GymBuddy” account.
- At least one workout exists in the "Workouts" section.

Test Steps:

1. Navigate to the **"Calendar"** section.
2. Click **"Schedule Workout"**.
3. Select an existing workout from the list.
4. Choose a **date and time** for the workout session.
5. Click **"Confirm"**.
6. Navigate back to the **"Calendar"** to verify that the workout is displayed on the selected date.

Test Cases:

Input Data	Expected Result	Comments
Selected Workout: "Leg Day"	Workout is scheduled successfully	Appears in the calendar
Date: 2025-03-15, Time: 10:00 AM	Date and time are displayed correctly	User receives notification if enabled
Recurrent Workout: Every Monday	Workout appears on all selected days	Repeat setting is saved correctly

---

Testing schedule

Testing task	Start	Deadline
Unit testing	2025-03-01	2025-04-01
Static testing	2025-04-01	2025-04-24
Integration testing	2025-04-24	2025-05-01
Performance testing	2025-05-01	2025-05-19
Usability testing	2025-05-19	2025-05-26
Acceptance testing	2025-05-26	2025-05-31