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

This document outlines the test plan for **“GymBuddy”,** a mobile app for Android and iOS. 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, and find workout partners based on location and preferences. The app includes guided exercises, progress tracking, and integration with fitness-related features to enhance the overall gym experience. It is being developed using **.NET MAUI** for cross-platform compatibility on **Android and iOS**.

Tests scope

Acceptance testing consists of:
“GymBuddy” mobile application (Android and iOS), 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.
- Integration Testing - Ensuring different parts of the app work together.
- Performance Testing - Assessing speed and stability.
- Usability Testing - Checking ease of use.
- Static Testing – Reviewing code, design, and documentation to identify defects early.

Prerequisites

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

1. Fully Functional App Prototype
   - The latest working version of “GymBuddy” must be available for testing on both Android and iOS.
   - Core features should be implemented.

2. Defined Test Cases
   - A list of structured test cases covering major functionalities.
   - Test cases should include expected inputs, steps, and expected outcomes.

3. Testing Environment Setup
   - Devices: Access to multiple smartphones (Android & iOS) for testing.
   - OS Versions: Ensure compatibility with Android 10+ and iOS 14+.
   - Network Conditions: A stable internet connection to test cloud-based features.
4. Bug Tracking System
   - A method for documenting bugs and issues.
   - Each issue should include steps to reproduce, screenshots, and priority level.
5. Dedicated Testers
   - At least 2-3 team members responsible for testing.
   - Testers should follow structured test cases and document findings.
6. Issue Resolution Plan
   - Developers should be available to fix bugs as they are reported.
   - A process should be in place for retesting fixed issues.
7. User Feedback Collection
   - Select a few external users (friends or course mates) for usability testing.
   - Gather feedback on app performance, design, and ease of use.

By ensuring these prerequisites are met, we can conduct structured and effective testing for “GymBuddy”.

Test priorities

Testing will focus on critical areas to ensure “GymBuddy” functions smoothly, provides a great user experience, and meets performance expectations. The main priorities are:

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 and operating systems. 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, third-party APIs, 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, including iOS, Android, 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

In order to fully test the app, several testing methods will be employed in order to ensure that the app is working as expected. Those include:

1. **Unit testing**
    - Will be used to verify the functionality of individual features and components. In order to achieve that, there will be test cases written for each function, method or class, and by executing the tests it will allow the tester to validate that the mobile application's code is working as intended or if it requires fixes.
2. **Integration testing**
    - Ensures that different modules of the mobile app work together as expected. It verifies interaction between components and helps identify issues that might not be seen by executing unit tests.
3. **Static analysis**
    - Method, which examines the application's source code without executing, in order to identify any potential issues beforehand, which might become apparent after launching the application.
4. **Performance testing**
    - In order to assess the responsiveness of the application, there will be several tests conducted:
        - Response time - will measure the time taken to respond to user actions.
        - Resource usage testing - evaluate the app's utilization of device resources, such as CPU, memory and battery.
        - Stability testing - ensure the app runs smoothly and without freezing for extended periods of time.
        - Stress testing - simulate heavy usage scenarios, where there are large amounts of data stored in the application.
5. **Usability testing**
    - The application will be tested in order to confirm whether the app is intuitive, easy to navigate and provides a positive user experience. In order to achieve that, we will try to gather feedback and make changes if necessary.

In order to streamline this process we will be using several testing techniques:

1. Scripted Test cases - scripted cases with predefined input and output data.
2. Exploratory Test Cases without data - the tester will choose the input data.
3. Usability evaluation - use of checklist to assess the ease of use of the application.
4. Performance data collection - gathering and comparing performance data in order to evaluate the responsiveness, stability and efficiency of the application.

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

Software that will be used for testing
Visual Studio 2022
xUnit (latest compatible version for .NET MAUI in Visual Studio 2022)
Roslyn Analyzers (built into Visual Studio 2022)
Visual Studio Profiler (part of Visual Studio 2022)

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

Test script 3: User Registration

Description: This script tests whether a new user can successfully register for the “GymBuddy” app.

Initial Data:

- The “GymBuddy” app is installed and opened.
- The user has not previously registered an account.

Test Steps:

1. Open the “GymBuddy” app.
2. Click on **"Sign Up"**.
3. Enter a **valid email, username, and password**.
4. Confirm the password.
5. Click **"Register"**.
6. Verify that a **confirmation email** is sent.
7. Click on the email verification link.
8. Return to the app and attempt to log in.

Test Cases:

Email	Username	Password	Expected Result	Comments
valid.email@example.com	user123	StrongPass1!	Registration successful	Confirmation email is sent
invalid-email	user456	StrongPass1!	Registration fails	Invalid email format
valid.email@example.com	user789	pass	Registration fails	Password does not meet security requirements

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