Kaunas technology university

Faculty of informatics

Team: Grupė 1
Project: “GymBuddy”

Software systems testing

LAB4 – Performance testing

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

Studentai

Eligijus Kiudys

Dėstytojai

Kaunas, 2025

Content

1. Introduction

1.1. Purpose of performance testing

The primary goal of performance testing in this project is to evaluate how efficiently the application utilizes system resources, particularly CPU and memory. By monitoring resource consumption during key user interactions, we can assess whether the app performs smoothly and identify areas where performance may degrade over time or with increased usage. Another important aspect is to observe how the app behaves under stress, such as rapid or repeated database operations, to ensure it remains stable and responsive. Through this process, we aim to detect performance bottlenecks, such as memory leaks, inefficient methods, or excessive CPU usage, and make improvements that enhance the overall user experience. This testing is especially important for our application, as mobile devices vary in performance capabilities, and ensuring the app runs reliably across different environments is critical to its usability and quality.

1.2. Overview of the app

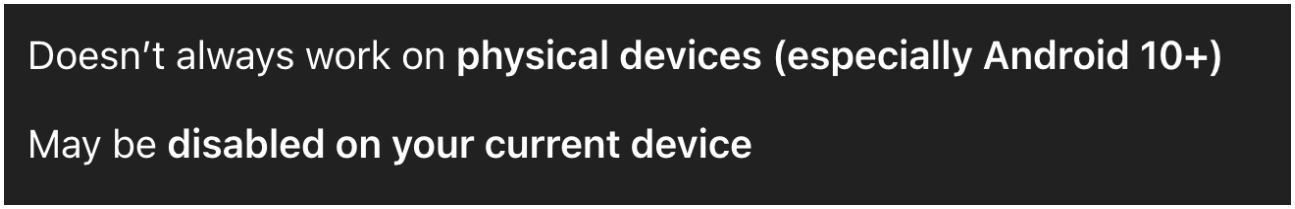
The application is a simple Android app developed using Android Studio, designed to manage and store user data locally on the device. It uses Room, a persistence library that provides an abstraction layer over SQLite, to handle all database operations such as creating, reading, updating, and deleting records. One of the key characteristics of the app is that it does not connect to the internet or use any form of cloud storage. All user data is stored locally on the device. This ensures data privacy and allows the app to function offline. Additionally, the app is designed for a single user, meaning there is no user authentication system or functionality to share data between users. This focus on simplicity and local functionality makes performance optimization of local resources especially important.

2. Test environment setup

2.1. Devices and emulator information

Hardware used for testing:

(personal devices didn’t work)



- Samsung Galaxy S21 (Android 14) (API Level 34)

Emulators used for testing:

- Medium Phone (API Level 35)

There are some negatives of running an emulator, because emulators run on a computer's hardware and have overhead. They do not perfectly replicate the CPU, GPU, and memory characteristics of real mobile devices. Performance on an emulator might be faster or slower than on a real device, leading to misleading performance test results or missed bottlenecks that only appear under real mobile hardware constraints.

These environments were used to install, run, and profile the app during performance testing to ensure consistent behavior across both physical and virtual devices.

2.2. Testing tools

Android Profiler (built into Android Studio): The Android Profiler is a suite of real-time profiling tools integrated directly into Android Studio. It provides live data about an app's resource usage while it's running on a connected device or emulator. It allows developers to observe the impact of their code and user interactions on the device's resources in real-time. You can perform actions within your app and immediately see how they affect CPU, memory, network, and energy consumption.

Heap Analyzer (via Android Studio's memory dump feature): it is a component of the Memory Profiler in Android Studio. It allows a developer to capture a snapshot of an app's memory at a specific point in time and then analyze the objects currently in memory. While the Memory Profiler shows memory usage over time, the Heap Analyzer provides a detailed view of the objects residing in the heap. This is crucial for identifying memory leaks, excessive object allocation and other operations related to memory.

Custom Kotlin Script: Used to simulate repeated database operations for local load testing in the absence of a REST API. It allows for highly specific and targeted load testing. Developers can write a script that directly interacts with your application's components, such as repeatedly performing database read/write operations, complex calculations, or other resource-intensive tasks.

3. Profiling test cases

3.1. Why is manual testing important?

While automated tests run scripts repeatedly, a manual tester can use the application realistically for an extended period to observe if performance degrades over time, if memory usage steadily climbs (indicating potential leaks not immediately obvious), or if battery drain is excessive during typical usage patterns, furthermore it allows the developer to see the user experience.

3.2. Manual test scenarios executed

Add new item to Room DB (adding new item)

Edit existing entry (editing existing entry)

Delete entry (deleting entry)

List/load all entries (opening a page with all entries)

Search/filter operations (executing search operation)

We repeated these tests three times in order to ensure that they provide the correct results and that the results do not vary by a significant margin.

3.3. CPU usage analysis

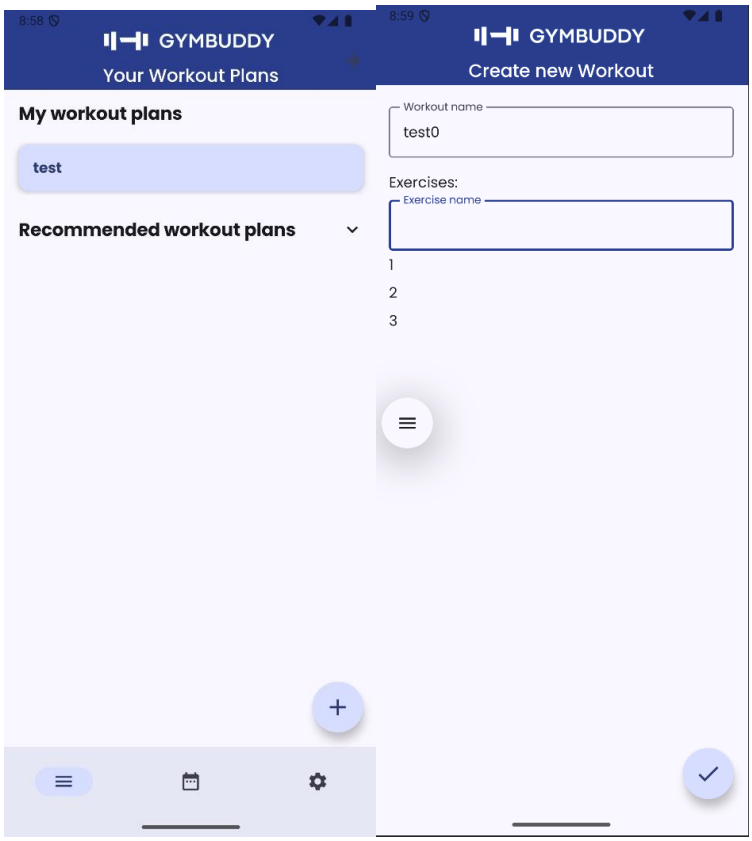
Method used: Android Profiler

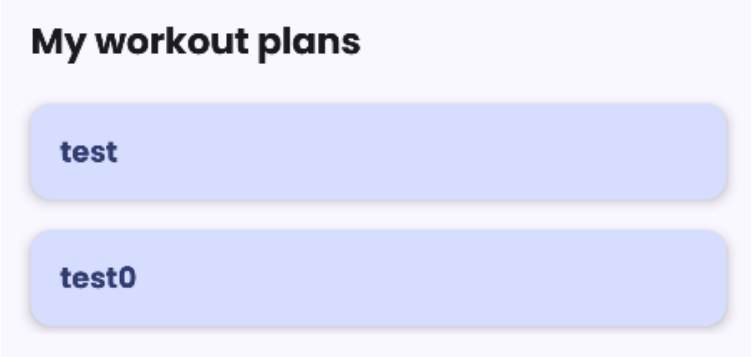
Observations:

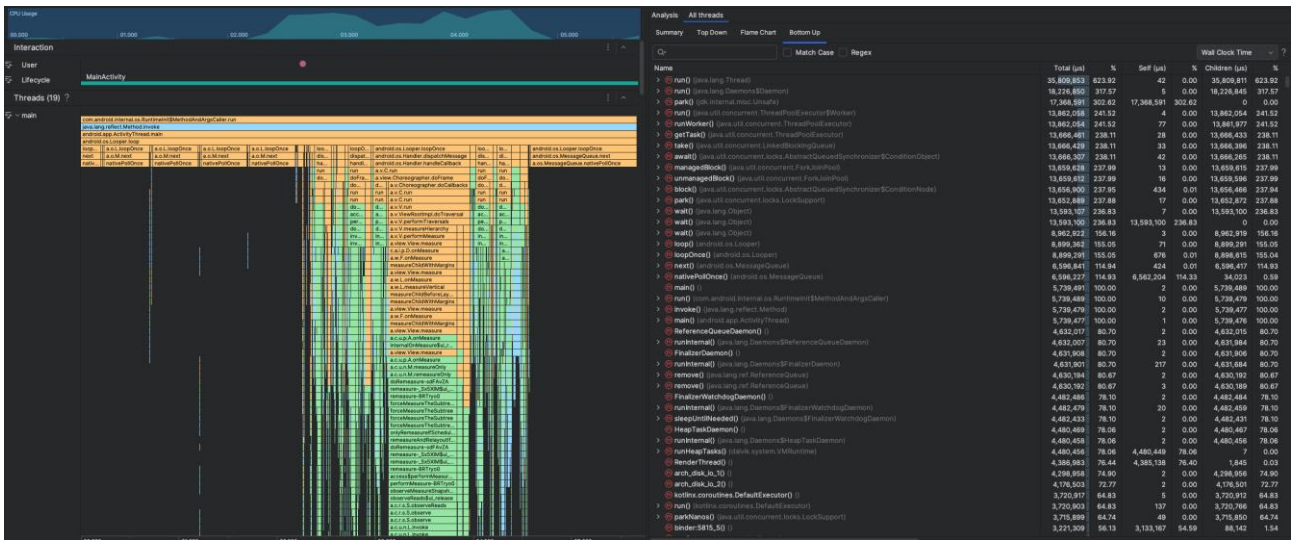
- Total CPU time during test
- Top methods/classes consuming CPU
- E.g.: DAO insert(), RecyclerView binding, LiveData updates

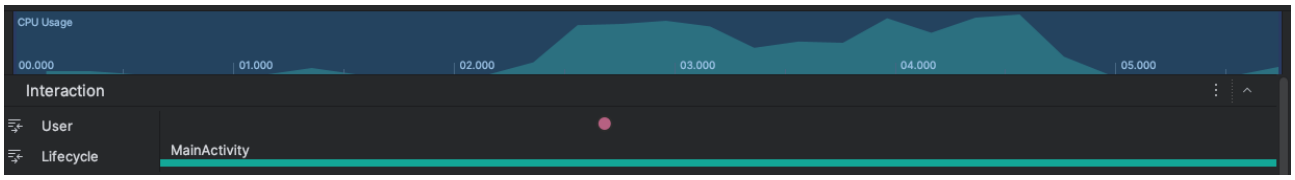
CPU usage and methods utilizing the most of the CPU:

Insert (used to test workout insertion into database operation):







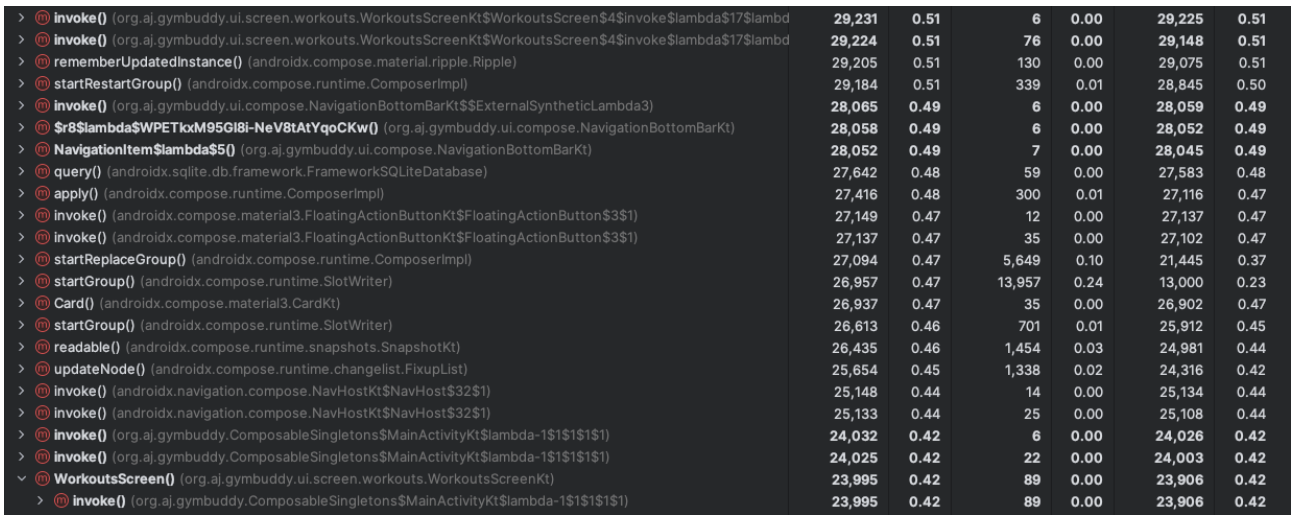


Analysis | All threads

Summary | Top Down | Flame Chart | Bottom Up

org.aj.gymbuddy | Match Case | Regex | Wall Clock Time

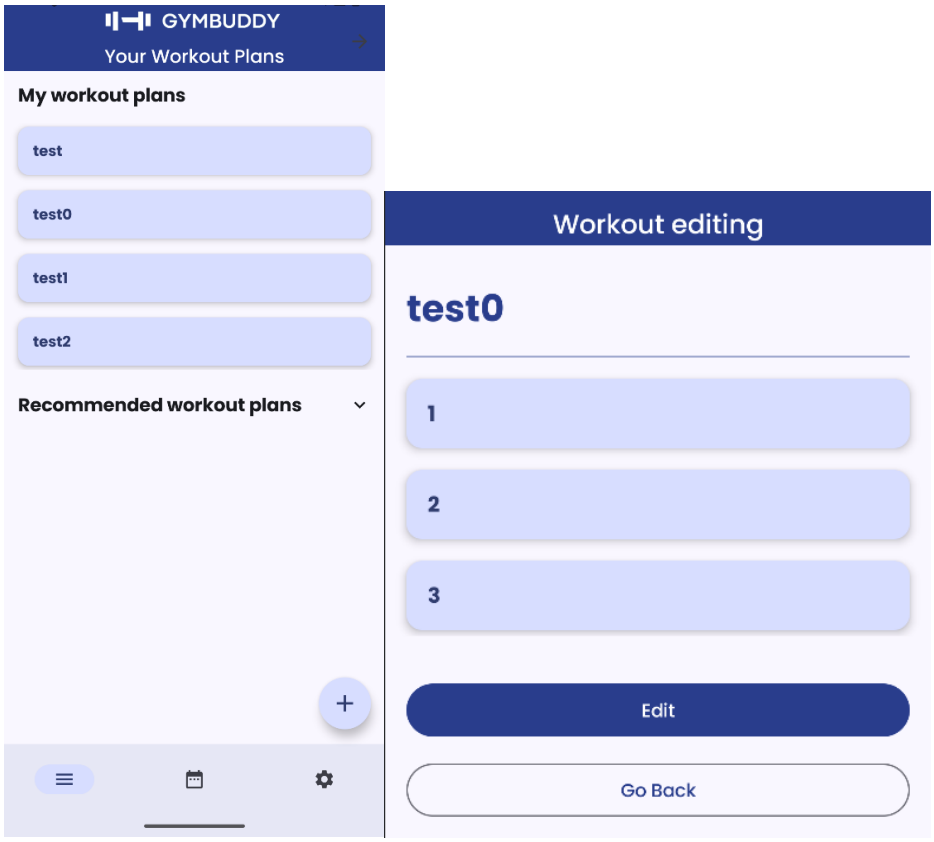
Name	Total (µs)	%	Self (µs)	%	Children (µs)	%
resumeWith() (kotlin.coroutines.jvm.internal.BaseContinuationImpl)	158,875	2.77	80	0.00	158,795	2.77
skipToGroupEnd() (androidx.compose.runtime.ComposerImpl)	156,316	2.72	97	0.00	156,219	2.72
TopBar() (org.aj.gymbuddy.ui.compose.TopBarKt)	149,953	2.61	130	0.00	149,823	2.61
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$1)	118,175	2.06	54	0.00	118,121	2.06
TopBar$lambda$9() (org.aj.gymbuddy.ui.compose.TopBarKt)	31,670	0.55	73	0.00	31,597	0.55
$r8$lambda$Zo_FBqabXSTQjnwlcQ1KvvMqkhI() (org.aj.gymbuddy.ui.compose.TopBarKt)	31,670	0.55	73	0.00	31,597	0.55
invoke() (org.aj.gymbuddy.ui.screen.workout.WorkoutScreenKt$WorkoutScreen$1)	108	0.00	3	0.00	105	0.00
end() (androidx.compose.runtime.ComposerImpl)	132,090	2.30	17,842	0.31	114,248	1.99
Transition-DTcfvLk() (androidx.compose.material3.TextFieldTransitionScope)	129,978	2.26	437	0.01	129,541	2.26
CenterAlignedTopAppBar() (androidx.compose.material3.AppBarKt)	122,654	2.14	24	0.00	122,630	2.14
invoke() (androidx.compose.runtime.internal.ComposableLambdaImpl$invoke$1)	119,038	2.07	12	0.00	119,026	2.07
invoke() (androidx.compose.runtime.internal.ComposableLambdaImpl$invoke$1)	119,024	2.07	11	0.00	119,013	2.07
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$1)	118,317	2.06	6	0.00	118,311	2.06
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$1)	118,311	2.06	4	0.00	118,307	2.06
NavigationItem() (org.aj.gymbuddy.ui.compose.NavigationBottomBarKt)	115,968	2.02	93	0.00	115,875	2.02
access$NavigationItem() (org.aj.gymbuddy.ui.compose.NavigationBottomBarKt)	87,923	1.53	44	0.00	87,879	1.53
invoke() (org.aj.gymbuddy.ui.compose.NavigationBottomBarKt$NavigationBottomBar$1)	87,923	1.53	44	0.00	87,879	1.53
invoke() (org.aj.gymbuddy.ui.compose.NavigationBottomBarKt$NavigationBottomBar$1)	87,923	1.53	44	0.00	87,879	1.53
NavigationItem$lambda$5() (org.aj.gymbuddy.ui.compose.NavigationBottomBarKt)	28,045	0.49	49	0.00	27,996	0.49
$r8$lambda$WPETkxM95GI8i-NeV8tAtYqoCKw() (org.aj.gymbuddy.ui.compose.NavigationBottomBarKt)	28,045	0.49	49	0.00	27,996	0.49
measure-3p2s80s() (androidx.compose.foundation.layout.PaddingValuesModifier)	111,338	1.94	34	0.00	111,304	1.94
invoke() (androidx.navigation.compose.NavHostKt$NavHost$34)	110,410	1.92	12	0.00	110,398	1.92
invoke() (androidx.navigation.compose.NavHostKt$NavHost$34)	110,397	1.92	10	0.00	110,387	1.92
NavHost() (androidx.navigation.compose.NavHostKt)	110,386	1.92	359	0.01	110,027	1.92
NavigationBarItem() (androidx.compose.material3.NavigationBarKt)	109,997	1.92	352	0.01	109,645	1.91
measure-3p2s80s() (androidx.compose.foundation.layout.PaddingNode)	108,779	1.90	7	0.00	108,772	1.90
measure-3p2s80s() (androidx.compose.foundation.layout.ColumnMeasurePolicy)	108,540	1.89	7	0.00	108,533	1.89
measure$default() (androidx.compose.foundation.layout.RowColumnMeasurePolicyKt)	108,520	1.89	1	0.00	108,519	1.89
measure() (androidx.compose.foundation.layout.RowColumnMeasurePolicyKt)	108,520	1.89	27	0.00	108,493	1.89
invoke() (androidx.compose.material3.ScaffoldKt$ScaffoldLayoutWithMeasureFix$1$1$bottomBarPlaceables$1)	107,093	1.87	2	0.00	107,091	1.87
invoke() (androidx.compose.material3.ScaffoldKt$ScaffoldLayoutWithMeasureFix$1$1$bottomBarPlaceables$1)	107,090	1.87	3	0.00	107,087	1.87
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$2)	106,559	1.86	5	0.00	106,554	1.86
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$2)	106,554	1.86	8	0.00	106,546	1.86
NavigationBottomBar() (org.aj.gymbuddy.ui.compose.NavigationBottomBarKt)	106,543	1.86	18	0.00	106,525	1.86
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$2)	106,543	1.86	18	0.00	106,525	1.86
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$2)	106,543	1.86	18	0.00	106,525	1.86
AnimatedContent() (androidx.compose.animation.AnimatedContentKt)	104,376	1.82	245	0.00	104,131	1.81

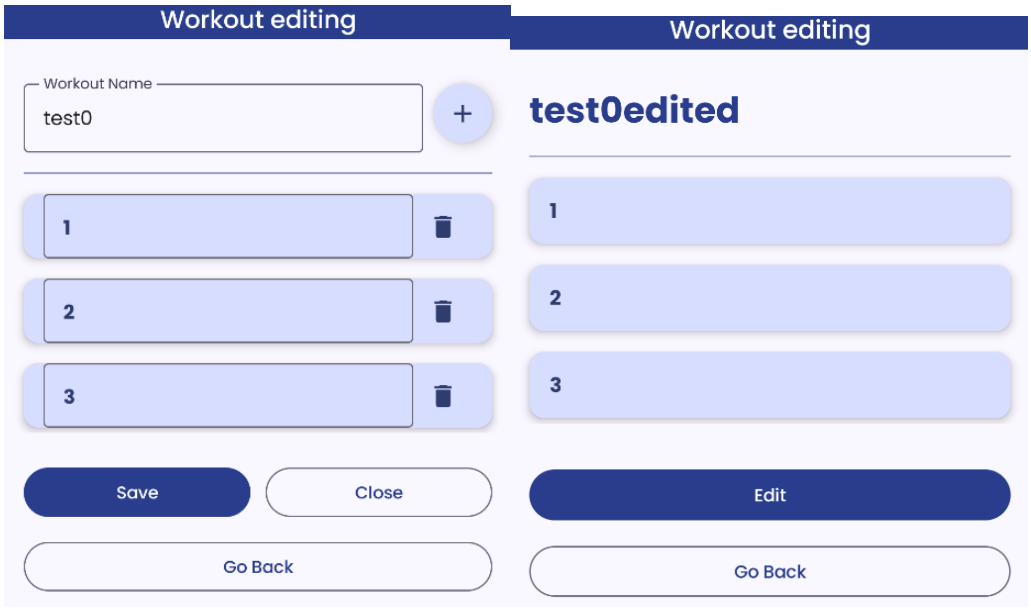


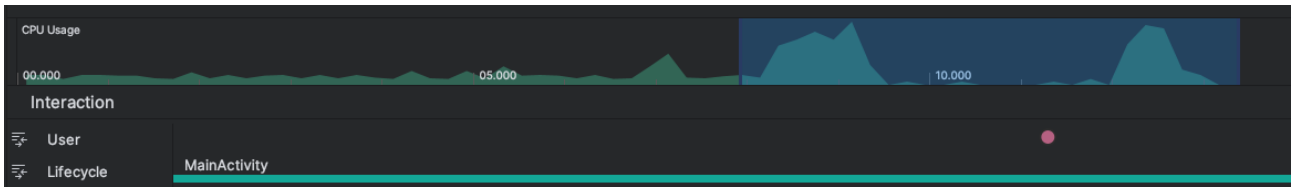
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$4$invoke$lambda17lambd	**29,231**	**0.51**	**6**	**0.00**	**29,225**	**0.51**
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$4$invoke$lambda17lambd	**29,224**	**0.51**	**76**	**0.00**	**29,148**	**0.51**
rememberUpdatedInstance() (androidx.compose.material.ripple.Ripple)	29,205	0.51	130	0.00	29,075	0.51
startRestartGroup() (androidx.compose.runtime.ComposerImpl)	29,184	0.51	339	0.01	28,845	0.50
invoke() (org.aj.gymbuddy.ui.compose.NavigationBottomBarKt$$ExternalSyntheticLambda3)	**28,065**	**0.49**	**6**	**0.00**	**28,059**	**0.49**
$r8$lambda$WPETkxM95Gl8i-NeV8tAtYqoCKw() (org.aj.gymbuddy.ui.compose.NavigationBottomBarKt)	**28,058**	**0.49**	**6**	**0.00**	**28,052**	**0.49**
NavigationItem$lambda$5() (org.aj.gymbuddy.ui.compose.NavigationBottomBarKt)	**28,052**	**0.49**	**7**	**0.00**	**28,045**	**0.49**
query() (androidx.sqlite.db.framework.FrameworkSQLiteDatabase)	27,642	0.48	59	0.00	27,583	0.48
apply() (androidx.compose.runtime.ComposerImpl)	27,416	0.48	300	0.01	27,116	0.47
invoke() (androidx.compose.material3.FloatingActionButtonKt$FloatingActionButton$3$1)	27,149	0.47	12	0.00	27,137	0.47
invoke() (androidx.compose.material3.FloatingActionButtonKt$FloatingActionButton$3$1)	27,137	0.47	35	0.00	27,102	0.47
startReplaceGroup() (androidx.compose.runtime.ComposerImpl)	27,094	0.47	5,649	0.10	21,445	0.37
startGroup() (androidx.compose.runtime.SlotWriter)	26,957	0.47	13,957	0.24	13,000	0.23
Card() (androidx.compose.material3.CardKt)	26,937	0.47	35	0.00	26,902	0.47
startGroup() (androidx.compose.runtime.SlotWriter)	26,613	0.46	701	0.01	25,912	0.45
readable() (androidx.compose.runtime.snapshots.SnapshotKt)	26,435	0.46	1,454	0.03	24,981	0.44
updateNode() (androidx.compose.runtime.changelist.FixupList)	25,654	0.45	1,338	0.02	24,316	0.42
invoke() (androidx.navigation.compose.NavHostKt$NavHost$32$1)	25,148	0.44	14	0.00	25,134	0.44
invoke() (androidx.navigation.compose.NavHostKt$NavHost$32$1)	25,133	0.44	25	0.00	25,108	0.44
invoke() (org.aj.gymbuddy.ComposableSingletons$MainActivityKt$lambda-1$1$1$1$1)	**24,032**	**0.42**	**6**	**0.00**	**24,026**	**0.42**
invoke() (org.aj.gymbuddy.ComposableSingletons$MainActivityKt$lambda-1$1$1$1$1)	**24,025**	**0.42**	**22**	**0.00**	**24,003**	**0.42**
WorkoutsScreen() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt)	**23,995**	**0.42**	**89**	**0.00**	**23,906**	**0.42**
invoke() (org.aj.gymbuddy.ComposableSingletons$MainActivityKt$lambda-1$1$1$1$1)	**23,995**	**0.42**	**89**	**0.00**	**23,906**	**0.42**

While inserting new items into the database, CPU usage spiked to 80-85%. Methods that used the most CPU: TopBar, invoke, NavigationItem.

Edit (used to test for editing an existing workout):





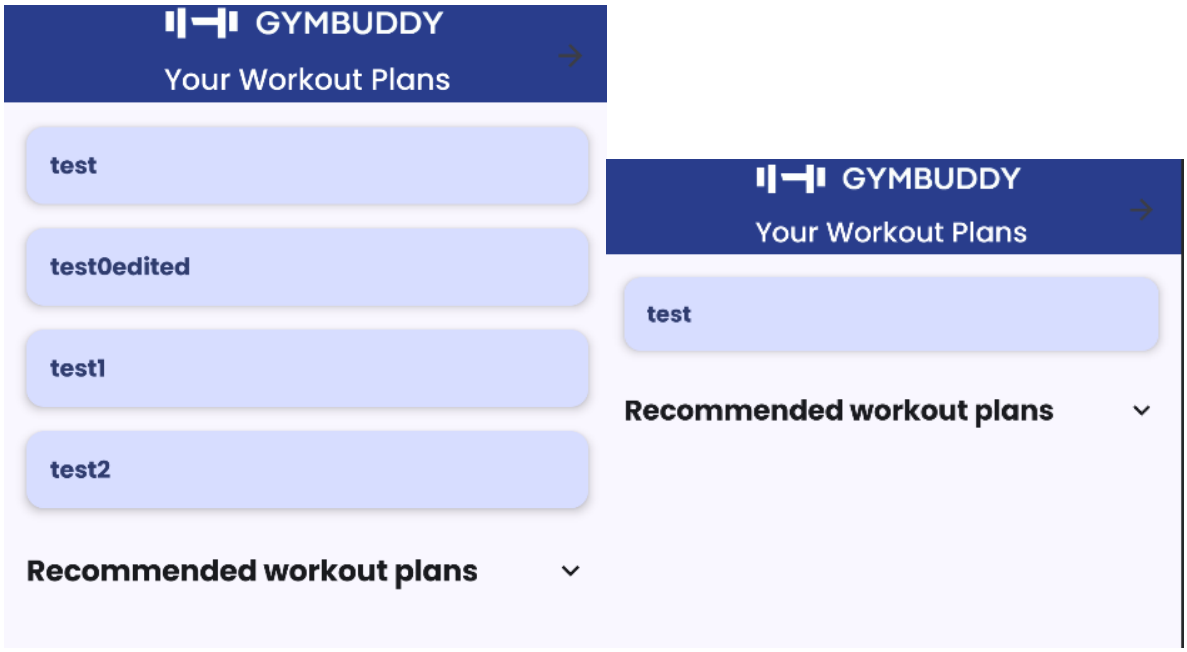


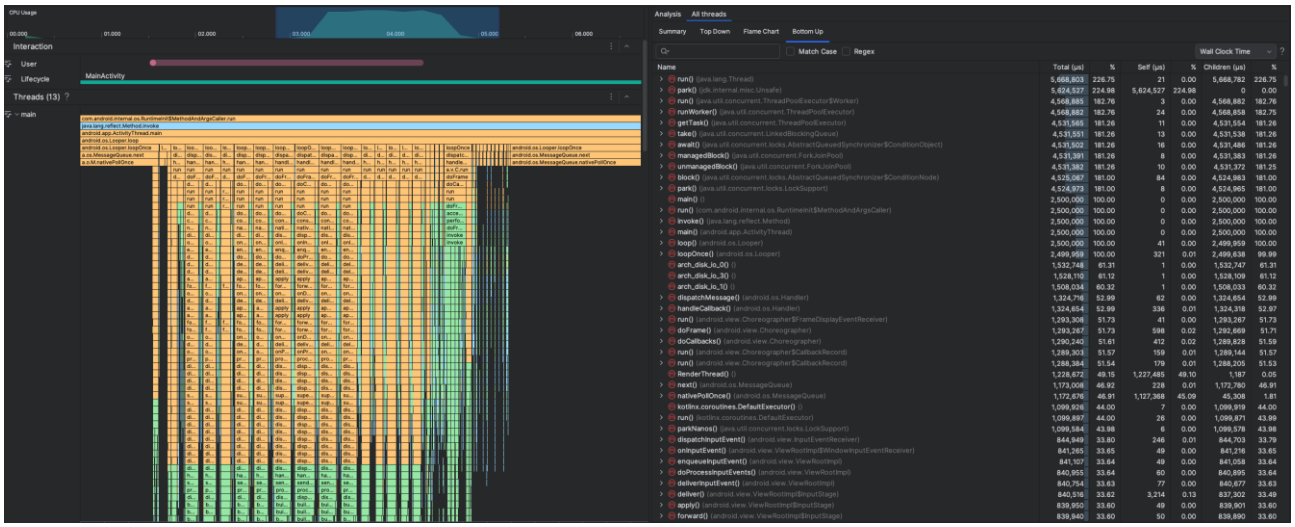
invoke() (org.aj.gymbuddy.ui.screen.workoutsedit.WorkoutsEditScreenKt$WorkoutsEditScreen$2$invoke$lambd	107,040	1.78	77	0.00	106,963	1.78
invoke() (androidx.compose.runtime.internal.ComposableLambdaImpl)	107,040	1.78	77	0.00	106,963	1.78
invoke() (org.aj.gymbuddy.ui.screen.workoutsedit.WorkoutsEditScreenKt$WorkoutsEditScreen$2$invoke$lambd	106,947	1.78	467	0.01	106,480	1.77
call() (org.aj.gymbuddy.db.WorkoutDao_Impl$8)	90,945	1.52	2	0.00	90,943	1.52
invokeSuspend() (androidx.room.CoroutinesRoom$Companion$createFlow$1$1$1)	90,945	1.52	2	0.00	90,943	1.52
call() (org.aj.gymbuddy.db.WorkoutDao_Impl$8)	90,943	1.52	30	0.00	90,913	1.52
call() (org.aj.gymbuddy.db.WorkoutDao_Impl$8)	90,943	1.52	30	0.00	90,913	1.52
Card() (androidx.compose.material3.CardKt)	86,145	1.44	134	0.00	86,011	1.43
measureAndLayout() (androidx.compose.ui.node.MeasureAndLayoutDelegate)	77,458	1.29	66	0.00	77,392	1.29
end() (androidx.compose.runtime.ComposerImpl)	71,868	1.20	9,747	0.16	62,121	1.04
invoke() (androidx.compose.runtime.internal.ComposableLambdaImpl)	71,128	1.19	9	0.00	71,119	1.19
invoke() (androidx.compose.runtime.internal.ComposableLambdaImpl)	71,117	1.19	38	0.00	71,079	1.18
invoke() (androidx.compose.material3.TextFieldImplKt$CommonDecorationBox$3)	70,365	1.17	25	0.00	70,340	1.17
invoke-eopBjH0() (androidx.compose.material3.TextFieldImplKt$CommonDecorationBox$3)	70,330	1.17	124	0.00	70,206	1.17
invoke() (org.aj.gymbuddy.ui.screen.workoutsedit.WorkoutsEditScreenKt$WorkoutsEditScreen$2$1$2$1$1$1)	63,627	1.06	73	0.00	63,554	1.06
invoke() (org.aj.gymbuddy.ui.screen.workoutsedit.WorkoutsEditScreenKt$WorkoutsEditScreen$2$1$2$1$1$1)	63,545	1.06	1,585	0.03	61,960	1.03
invoke() (org.aj.gymbuddy.ui.screen.workoutsedit.WorkoutsEditScreenKt$WorkoutsEditScreen$2$1$2$1$1$1)	63,545	1.06	1,585	0.03	61,960	1.03
run() (androidx.room.TransactionExecutor$$ExternalSyntheticLambda0)	61,366	1.02	3	0.00	61,363	1.02

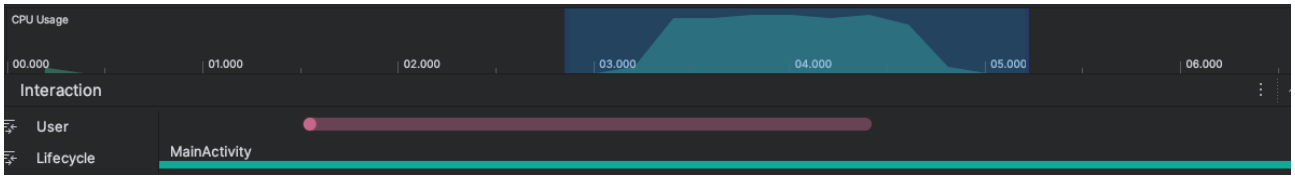
invoke() (org.aj.gymbuddy.db.WorkoutDao_Impl$$ExternalSyntheticLambda0)	11,192	0.19	4	0.00	11,188	0.19
lambda$upsert$0$org-aj-gymbuddy-db-WorkoutDao_Impl() (org.aj.gymbuddy.db.WorkoutDao_Impl)	11,188	0.19	1	0.00	11,187	0.19
upsert() (org.aj.gymbuddy.db.WorkoutDao$DefaultImpls)	11,187	0.19	8	0.00	11,179	0.19
lambda$upsert$0$org-aj-gymbuddy-db-WorkoutDao_Impl() (org.aj.gymbuddy.db.WorkoutDao_Impl)	11,187	0.19	8	0.00	11,179	0.19
update() (org.aj.gymbuddy.db.WorkoutDao_Impl)	11,166	0.19	3	0.00	11,163	0.19
upsert() (org.aj.gymbuddy.db.WorkoutDao$DefaultImpls)	11,166	0.19	3	0.00	11,163	0.19
lambda$upsert$0$org-aj-gymbuddy-db-WorkoutDao_Impl() (org.aj.gymbuddy.db.WorkoutDao_Impl)	11,166	0.19	3	0.00	11,163	0.19
execute() (androidx.room.CoroutinesRoom)	11,161	0.19	6	0.00	11,155	0.19
execute() (androidx.room.CoroutinesRoom$Companion)	11,155	0.19	5	0.00	11,150	0.19
invoke() (androidx.compose.foundation.text.BasicTextFieldKt$BasicTextField$8$1)	11,098	0.18	4	0.00	11,094	0.18
invoke() (androidx.compose.foundation.text.BasicTextFieldKt$BasicTextField$8$1)	11,094	0.18	44	0.00	11,050	0.18
call() (org.aj.gymbuddy.db.WorkoutDao_Impl$6)	11,083	0.18	2	0.00	11,081	0.18
call() (org.aj.gymbuddy.db.WorkoutDao_Impl$6)	11,081	0.18	4	0.00	11,077	0.18
handle() (androidx.room.EntityDeletionOrUpdateAdapter)	10,841	0.18	8	0.00	10,833	0.18
nativeRegisterLocalizedCollators() (android.database.sqlite.SQLiteConnection)	10,774	0.18	10,774	0.18	0	0.00
bind() (org.aj.gymbuddy.db.WorkoutDao_Impl$3)	10,627	0.18	1	0.00	10,626	0.18
handle() (androidx.room.EntityDeletionOrUpdateAdapter)	10,627	0.18	1	0.00	10,626	0.18
bind() (org.aj.gymbuddy.db.WorkoutDao_Impl$3)	10,626	0.18	21	0.00	10,605	0.18
bind() (org.aj.gymbuddy.db.WorkoutDao_Impl$3)	10,626	0.18	21	0.00	10,605	0.18
append() (java.lang.StringBuilder)	10,461	0.17	18	0.00	10,443	0.17
toString() (org.aj.gymbuddy.db.Converters)	10,459	0.17	1	0.00	10,458	0.17

While editing a workout item, two spikes were experienced, because in one text was being edited and in another the edited text was being saved. While editing, CPU usage spiked to 59-80% and upon saving the edited item it spiked to 61-90%. Methods that consumed the most CPU power: invoke(), call, upsert, update (methods related to the database operations of editing).

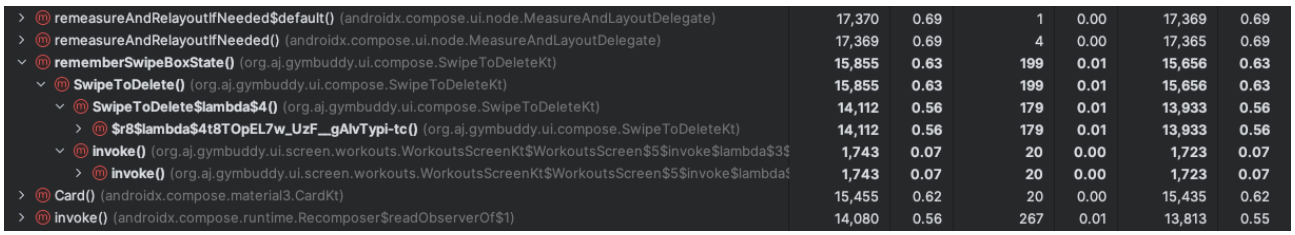
Delete:







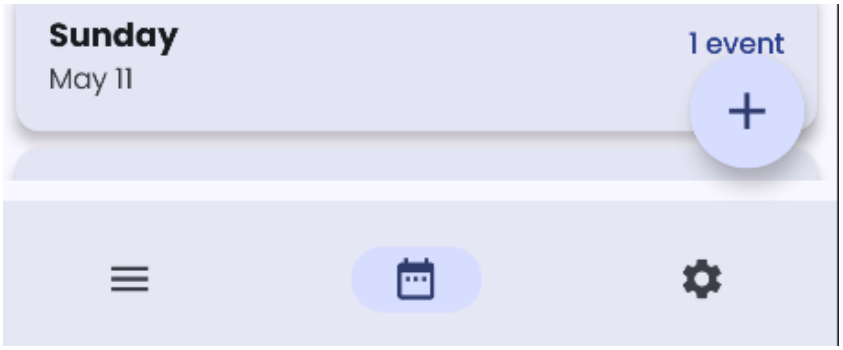
compose() (androidx.compose.runtime.RecomposeScopeImpl)	139,018	5.56	65	0.00	138,953	5.56
SwipeToDelete() (org.aj.gymbuddy.ui.compose.SwipeToDeleteKt)	110,104	4.40	411	0.02	109,693	4.39
SwipeToDelete$lambda$4() (org.aj.gymbuddy.ui.compose.SwipeToDeleteKt)	81,985	3.28	374	0.01	81,611	3.26
$r8$lambda$4t8TOpEL7w_UzF__gAlvTypi-tc() (org.aj.gymbuddy.ui.compose.SwipeToDeleteKt)	81,985	3.28	374	0.01	81,611	3.26
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$5$invoke$lambda3lan	28,119	1.12	37	0.00	28,082	1.12
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$5$invoke$lambda3	28,119	1.12	37	0.00	28,082	1.12
invoke() (org.aj.gymbuddy.ui.compose.SwipeToDeleteKt$$ExternalSyntheticLambda1)	82,142	3.29	70	0.00	82,072	3.28
$r8$lambda$4t8TOpEL7w_UzF__gAlvTypi-tc() (org.aj.gymbuddy.ui.compose.SwipeToDeleteKt)	82,070	3.28	27	0.00	82,043	3.28
invoke() (org.aj.gymbuddy.ui.compose.SwipeToDeleteKt$$ExternalSyntheticLambda1)	82,070	3.28	27	0.00	82,043	3.28
SwipeToDelete$lambda$4() (org.aj.gymbuddy.ui.compose.SwipeToDeleteKt)	82,043	3.28	52	0.00	81,991	3.28
$r8$lambda$4t8TOpEL7w_UzF__gAlvTypi-tc() (org.aj.gymbuddy.ui.compose.SwipeToDeleteKt)	82,043	3.28	52	0.00	81,991	3.28
SwipeToDismissBox() (androidx.compose.material3.SwipeToDismissBoxKt)	72,704	2.91	1,119	0.04	71,585	2.86

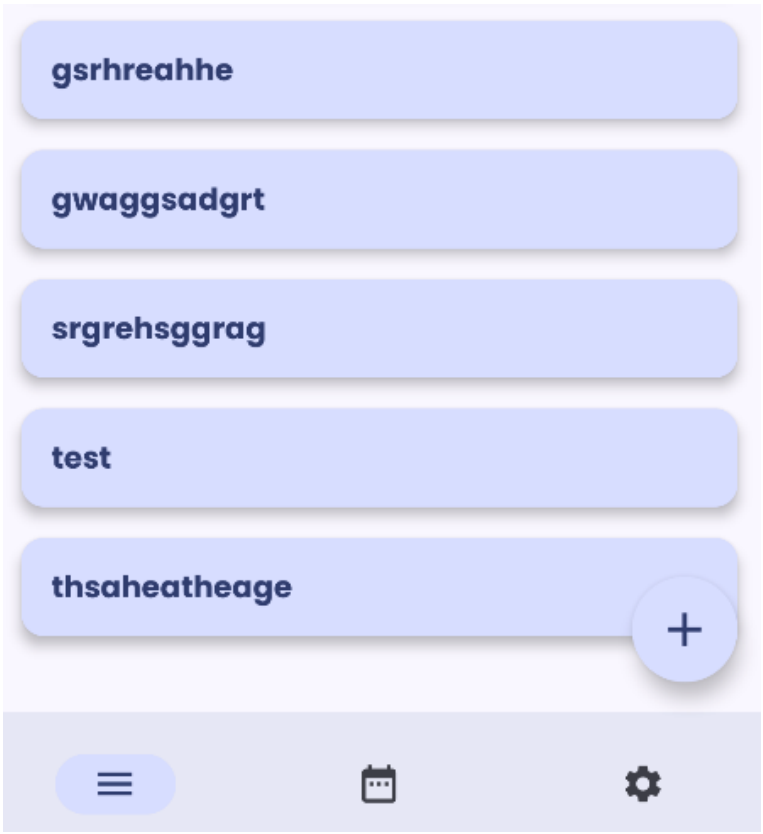


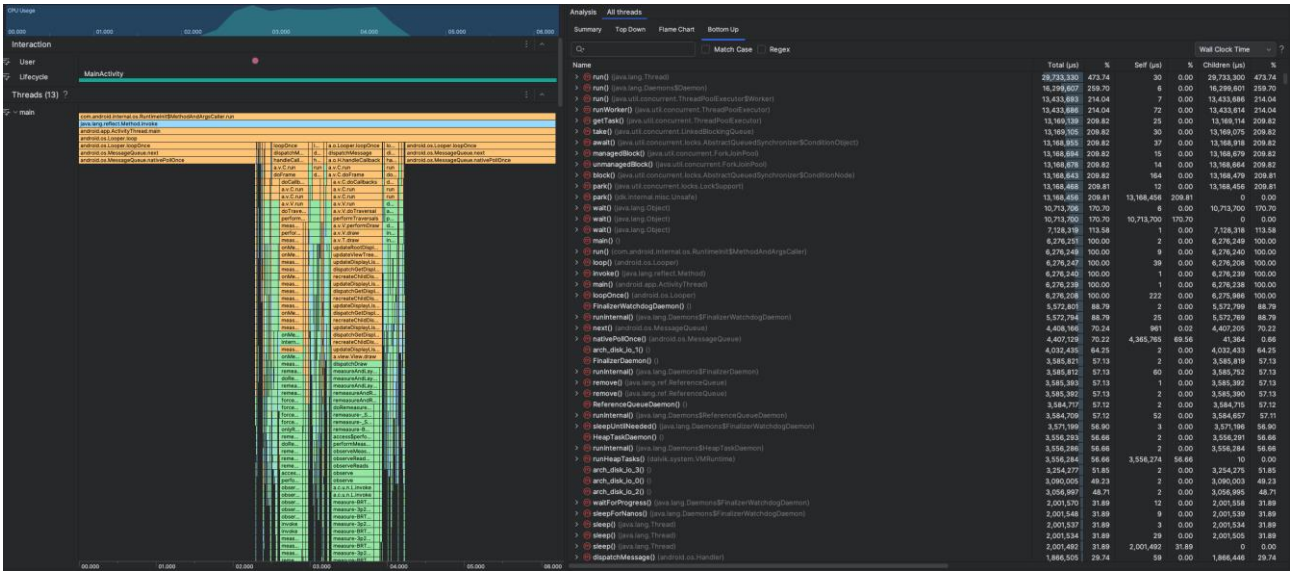
remeasureAndRelayoutIfNeeded$default() (androidx.compose.ui.node.MeasureAndLayoutDelegate)	17,370	0.69	1	0.00	17,369	0.69
remeasureAndRelayoutIfNeeded() (androidx.compose.ui.node.MeasureAndLayoutDelegate)	17,369	0.69	4	0.00	17,365	0.69
rememberSwipeBoxState() (org.aj.gymbuddy.ui.compose.SwipeToDeleteKt)	15,855	0.63	199	0.01	15,656	0.63
SwipeToDelete() (org.aj.gymbuddy.ui.compose.SwipeToDeleteKt)	15,855	0.63	199	0.01	15,656	0.63
SwipeToDelete$lambda$4() (org.aj.gymbuddy.ui.compose.SwipeToDeleteKt)	14,112	0.56	179	0.01	13,933	0.56
$r8$lambda$4t8TOpEL7w_UzF__gAlvTypi-tc() (org.aj.gymbuddy.ui.compose.SwipeToDeleteKt)	14,112	0.56	179	0.01	13,933	0.56
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$5$invoke$lambda3	1,743	0.07	20	0.00	1,723	0.07
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$5$invoke$lambda$	1,743	0.07	20	0.00	1,723	0.07
Card() (androidx.compose.material3.CardKt)	15,455	0.62	20	0.00	15,435	0.62
invoke() (androidx.compose.runtime.Recomposer$readObserverOf$1)	14,080	0.56	267	0.01	13,813	0.55

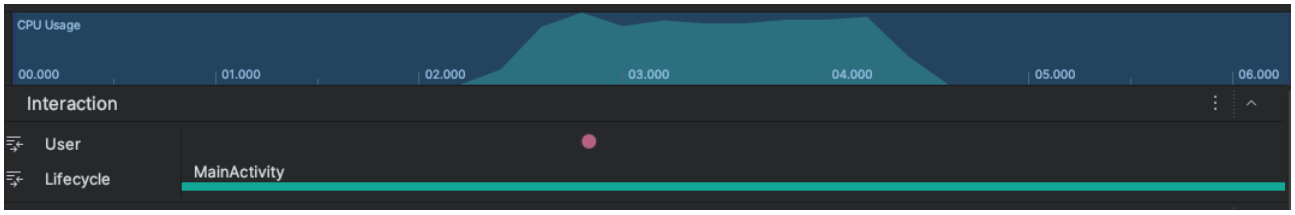
While deleting an item, CPU usage spiked to 80-90%. Methods that attributed the most to the spike: SwipeToDelete, invoke, rememberSwipeBoxState. The higher CPU usage is expected, because in most cases insertion takes less time and uses less power than deletion.

Loading and displaying a list:





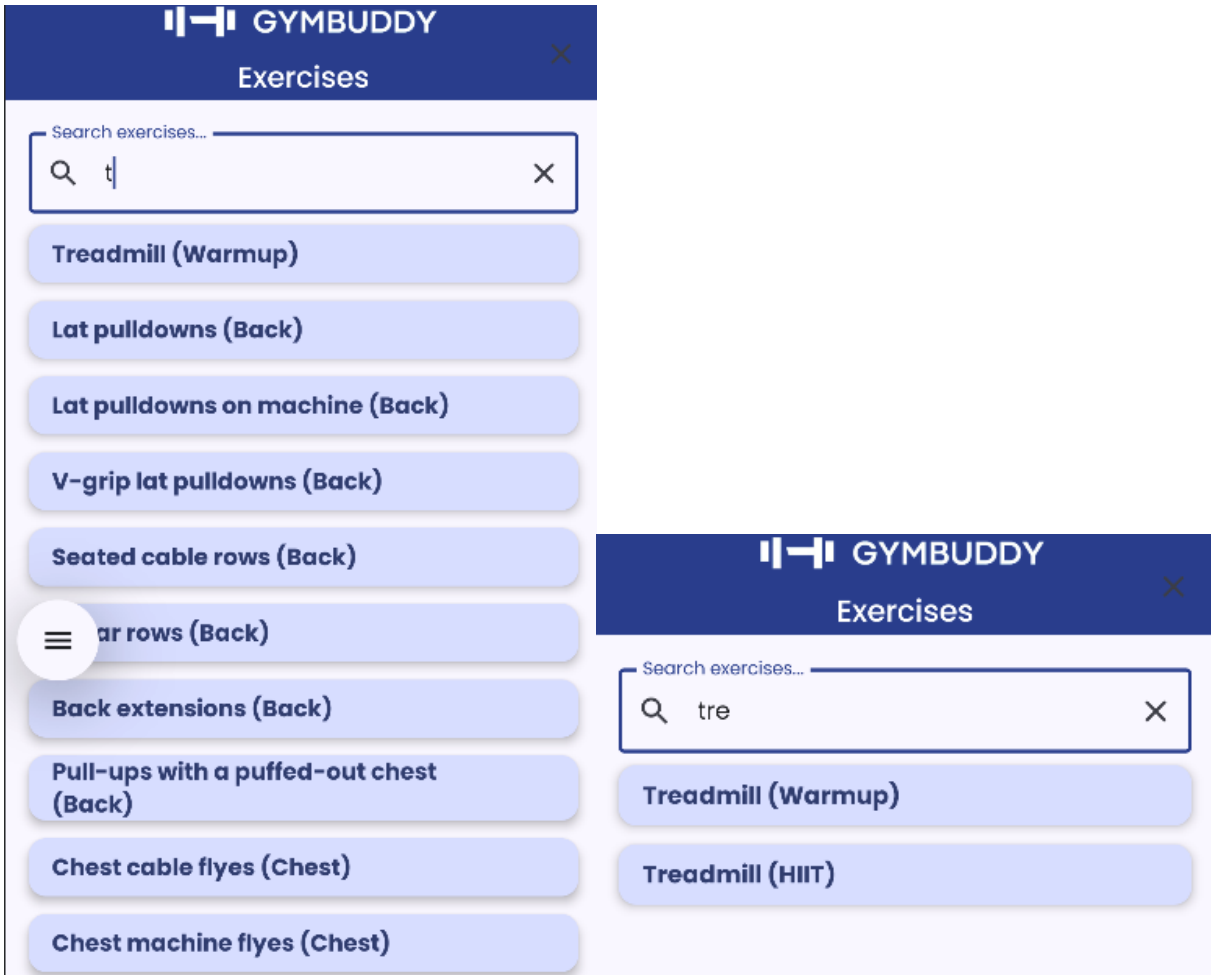


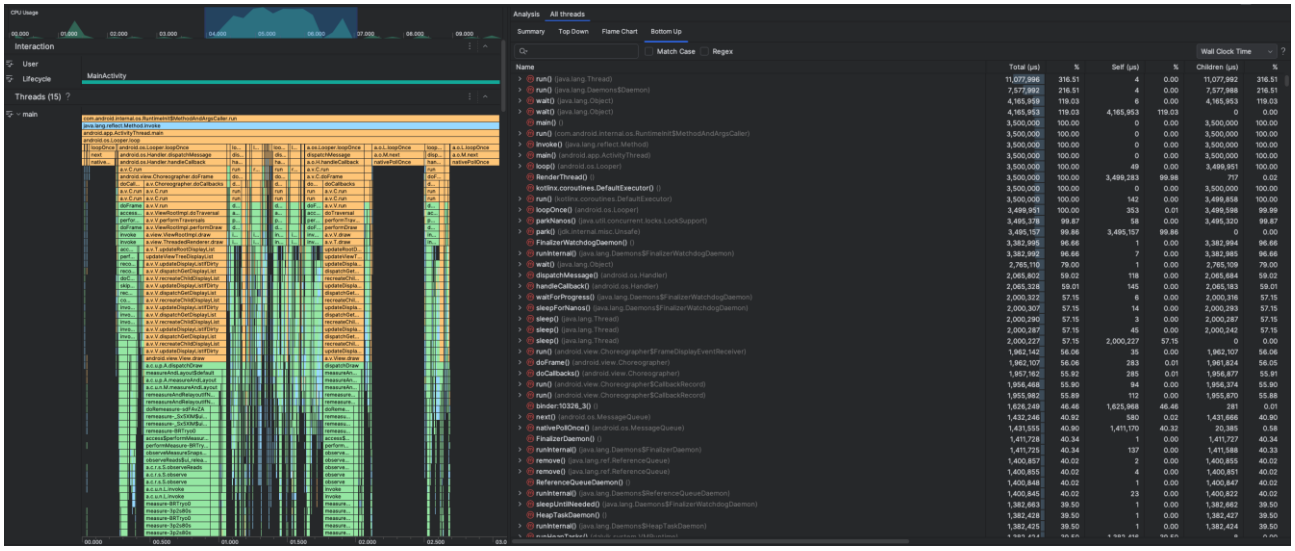


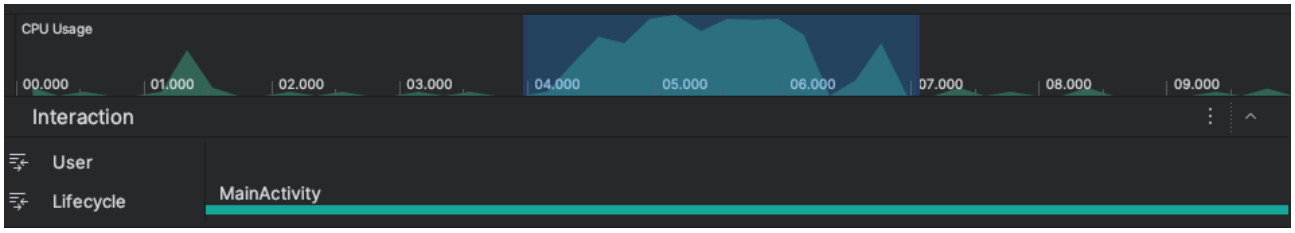
invoke() (org.aj.gymbuddy.ui.compose.NavigationBottomBarKt$$ExternalSyntheticLambda3)	77,885	1.24	13	0.00	77,872	1.24
$r8$lambda$WPETkxM95GI8i-NeV8tAtYqoCKw() (org.aj.gymbuddy.ui.compose.NavigationBottomBarKt)	77,872	1.24	20	0.00	77,852	1.24
NavigationItem$lambda$5() (org.aj.gymbuddy.ui.compose.NavigationBottomBarKt)	77,852	1.24	13	0.00	77,839	1.24
$r8$lambda$WPETkxM95GI8i-NeV8tAtYqoCKw() (org.aj.gymbuddy.ui.compose.NavigationBottomBarKt)	77,852	1.24	13	0.00	77,839	1.24
TopBar() (org.aj.gymbuddy.ui.compose.TopBarKt)	74,311	1.18	97	0.00	74,214	1.18
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$2)	50,074	0.80	46	0.00	50,028	0.80
TopBar$lambda$9() (org.aj.gymbuddy.ui.compose.TopBarKt)	24,237	0.39	51	0.00	24,186	0.39
invoke() (androidx.compose.ui.layout.LayoutKt$materializerOf$1)	72,954	1.16	163	0.00	72,791	1.16
invoke-Deg8D_g() (androidx.compose.ui.layout.LayoutKt$materializerOf$1)	72,737	1.16	594	0.01	72,143	1.15
run() (androidx.compose.ui.platform.AndroidUiDispatcher$dispatchCallback$1)	71,394	1.14	25	0.00	71,369	1.14
access$performTrampolineDispatch() (androidx.compose.ui.platform.AndroidUiDispatcher)	71,369	1.14	8	0.00	71,361	1.14
performTrampolineDispatch() (androidx.compose.ui.platform.AndroidUiDispatcher)	71,361	1.14	77	0.00	71,284	1.14
call() (org.aj.gymbuddy.db.SettingsDao_Impl$8)	70,191	1.12	1	0.00	70,190	1.12
call() (org.aj.gymbuddy.db.SettingsDao_Impl$8)	70,190	1.12	54	0.00	70,136	1.12
CenterAlignedTopAppBar() (androidx.compose.material3.AppBarKt)	68,821	1.10	19	0.00	68,802	1.10
invoke() (androidx.compose.material3.ScaffoldKt$ScaffoldLayoutWithMeasureFix$1$1$bottomBarPlaceables$1)	67,655	1.08	1	0.00	67,654	1.08
invoke() (androidx.compose.material3.ScaffoldKt$ScaffoldLayoutWithMeasureFix$1$1$bottomBarPlaceables$1)	67,653	1.08	15	0.00	67,638	1.08
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$3)	66,730	1.06	6	0.00	66,724	1.06
invoke() (org.aj.gymbuddy.ui.screen.workouts.WorkoutsScreenKt$WorkoutsScreen$3)	66,724	1.06	6	0.00	66,718	1.06
NavigationBottomBar() (org.aj.gymbuddy.ui.compose.NavigationBottomBarKt)	66,714	1.06	8	0.00	66,706	1.06
NavigationBar-HsRjFd4() (androidx.compose.material3.NavigationBarKt)	66,553	1.06	11	0.00	66,542	1.06

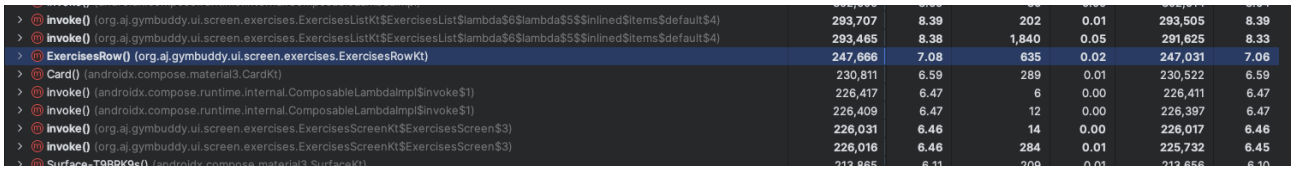
While loading lists, CPU usage spiked to 80-100%, because making a bigger list and emulating it takes longer. Methods that contributed to the spike: invoke, NavigationItem, TopBar, call, NavigationBottomBar.

Execute search:

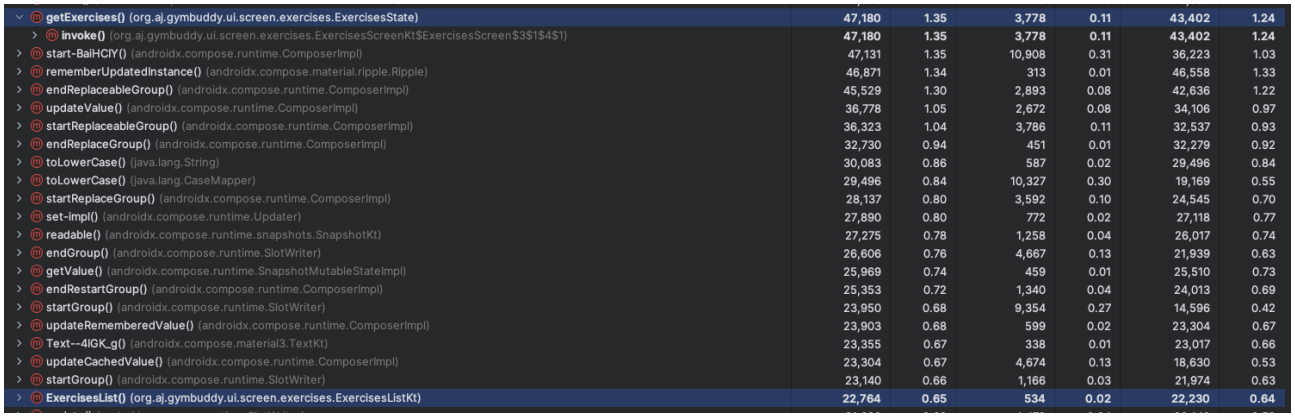








invoke() (org.aj.gymbuddy.ui.screen.exercises.ExercisesListKt$ExercisesList$lambda6lambda$5$$inlined$items$default$4)	293,707	8.39	202	0.01	293,505	8.39
invoke() (org.aj.gymbuddy.ui.screen.exercises.ExercisesListKt$ExercisesList$lambda6lambda$5$$inlined$items$default$4)	293,465	8.38	1,840	0.05	291,625	8.33
ExercisesRow() (org.aj.gymbuddy.ui.screen.exercises.ExercisesRowKt)	247,666	7.08	635	0.02	247,031	7.06
Card() (androidx.compose.material3.CardKt)	230,811	6.59	289	0.01	230,522	6.59
invoke() (androidx.compose.runtime.internal.ComposableLambdaImpl$invoke$1)	226,417	6.47	6	0.00	226,411	6.47
invoke() (androidx.compose.runtime.internal.ComposableLambdaImpl$invoke$1)	226,409	6.47	12	0.00	226,397	6.47
invoke() (org.aj.gymbuddy.ui.screen.exercises.ExercisesScreenKt$ExercisesScreen$3)	226,031	6.46	14	0.00	226,017	6.46
invoke() (org.aj.gymbuddy.ui.screen.exercises.ExercisesScreenKt$ExercisesScreen$3)	226,016	6.46	284	0.01	225,732	6.45



getExercises() (org.aj.gymbuddy.ui.screen.exercises.ExercisesState)	47,180	1.35	3,778	0.11	43,402	1.24
invoke() (org.aj.gymbuddy.ui.screen.exercises.ExercisesScreenKt$ExercisesScreen$3$1$4$1)	47,180	1.35	3,778	0.11	43,402	1.24
start-BaiHCIY() (androidx.compose.runtime.ComposerImpl)	47,131	1.35	10,908	0.31	36,223	1.03
rememberUpdatedInstance() (androidx.compose.material.ripple.Ripple)	46,871	1.34	313	0.01	46,558	1.33
endReplaceableGroup() (androidx.compose.runtime.ComposerImpl)	45,529	1.30	2,893	0.08	42,636	1.22
updateValue() (androidx.compose.runtime.ComposerImpl)	36,778	1.05	2,672	0.08	34,106	0.97
startReplaceableGroup() (androidx.compose.runtime.ComposerImpl)	36,323	1.04	3,786	0.11	32,537	0.93
endReplaceGroup() (androidx.compose.runtime.ComposerImpl)	32,730	0.94	451	0.01	32,279	0.92
toLowerCase() (java.lang.String)	30,083	0.86	587	0.02	29,496	0.84
toLowerCase() (java.lang.CaseMapper)	29,496	0.84	10,327	0.30	19,169	0.55
startReplaceGroup() (androidx.compose.runtime.ComposerImpl)	28,137	0.80	3,592	0.10	24,545	0.70
set-impl() (androidx.compose.runtime.Updater)	27,890	0.80	772	0.02	27,118	0.77
readable() (androidx.compose.runtime.snapshots.SnapshotKt)	27,275	0.78	1,258	0.04	26,017	0.74
endGroup() (androidx.compose.runtime.SlotWriter)	26,606	0.76	4,667	0.13	21,939	0.63
getValue() (androidx.compose.runtime.SnapshotMutableStateImpl)	25,969	0.74	459	0.01	25,510	0.73
endRestartGroup() (androidx.compose.runtime.ComposerImpl)	25,353	0.72	1,340	0.04	24,013	0.69
startGroup() (androidx.compose.runtime.SlotWriter)	23,950	0.68	9,354	0.27	14,596	0.42
updateRememberedValue() (androidx.compose.runtime.ComposerImpl)	23,903	0.68	599	0.02	23,304	0.67
Text--4IGK_g() (androidx.compose.material3.TextKt)	23,355	0.67	338	0.01	23,017	0.66
updateCachedValue() (androidx.compose.runtime.ComposerImpl)	23,304	0.67	4,674	0.13	18,630	0.53
startGroup() (androidx.compose.runtime.SlotWriter)	23,140	0.66	1,166	0.03	21,974	0.63
ExercisesList() (org.aj.gymbuddy.ui.screen.exercises.ExercisesListKt)	22,764	0.65	534	0.02	22,230	0.64

When executing the first search, there was a bigger list presented for the results. CPU usage spiked 90-100% and when continuing the search, CPU usage did not exceed the initial loading phase and hovered around 90-95%. Methods that were the cause of CPU usage spikes: invoke, ExercisesRow, getExercises, ExercisesList.

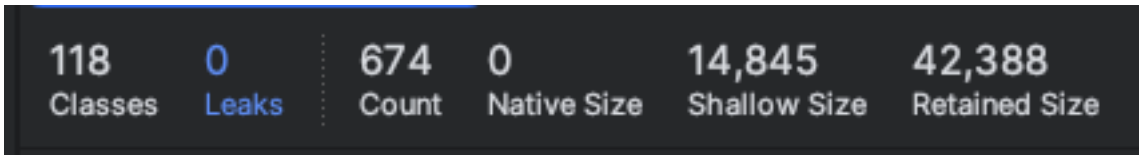
All of these tests show a high CPU usage, however that only happened, when computationally intensive tasks were performed that were related to the database. Furthermore, it is normal for an emulator to display high CPU usage, it does not mean that the program is unoptimized.

3.4. Memory usage analysis

Method used: Heap dump + analysis

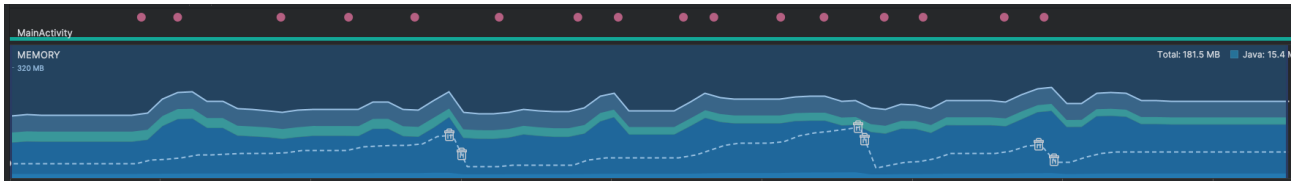
Findings:

- Most allocated objects (e.g., entity classes, strings)
- Memory leaks (if any)



Most allocations were made for the Exercise categories, however this is normal, because our app has a lot of workouts as well as exercises that are coded in.

While performing this analysis, there were no memory leaks found as can be seen in the provided results. Memory leaks can be extremely detrimental to a program's performance, because the longer an application runs the more memory it starts to use and the time it can take for UI as well as database operations can increase and the application performance degrades.



While performing database operations such as insert, delete, edit, save, load list and search there were several memory spikes, however as can be seen in the picture above, the spikes did not last long and quickly fell, thus the application’s performance did not degrade and performance impact was negligible.

4. Load/Stress testing

4.1. Simulated load strategy

Our application doesn't have a REST API, so we simulated load by writing a custom script that would perform these operations:

- Creating 1000/5000/10000/25000/50000 insert/read operations over a maximum of 5 minutes
- Using Kotlin coroutines or multi-threading

4.2. Results

Load test results at 1000 operations:

```
Robolectric Load Test Finished in 1.24 seconds.
Total Operations Attempted: 1000
Successful Operations: 1000
Failed Operations: 0
Response Time (for successful operations):
  Average: 578.87 ms
  Min: 38 ms
  Max: 904 ms
Throughput (successful ops/sec): 809.72
```

Results at 5000 operations:

```
Robolectric Load Test Finished in 3.68 seconds.
Total Operations Attempted: 5000
Successful Operations: 5000
Failed Operations: 0
Response Time (for successful operations):
  Average: 2037.04 ms
  Min: 206 ms
  Max: 3538 ms
Throughput (successful ops/sec): 1360.17
```

Results at 10000 operations:

```
Robolectric Load Test Finished in 6.16 seconds.
Total Operations Attempted: 10000
Successful Operations: 10000
Failed Operations: 0
Response Time (for successful operations):
  Average: 3390.46 ms
  Min: 204 ms
  Max: 6024 ms
Throughput (successful ops/sec): 1622.59
```

Operation results at 25000 insert/read operations:

```
Robolectric Load Test Finished in 16.58 seconds.
Total Operations Attempted: 25000
Successful Operations: 25000
Failed Operations: 0
Response Time (for successful operations):
  Average: 8237.40 ms
  Min: 277 ms
  Max: 16334 ms
Throughput (successful ops/sec): 1507.57
```

When the operations are 50000 the results are:

```
Robolectric Load Test Finished in 30.39 seconds.
Total Operations Attempted: 50000
Successful Operations: 50000
Failed Operations: 0
Response Time (for successful operations):
  Average: 16125.57 ms
  Min: 333 ms
  Max: 30033 ms
Throughput (successful ops/sec): 1645.39
```

Number of operations	Average, ms	Minimum, ms	Maximum, ms
1000	578	38	904
5000	2037	206	3538
10000	3390	204	6024
25000	8237	277	16344
50000	16125	333	30033

From the provided pictures and table, we can see that the time on average that it takes to complete an operation increases proportionately with the number of insert/read operations done to the database, which means that there might be issues if our app was to be scaled to hold millions of rows of data. However, as it stands, our application is primarily meant to be used for workouts and all of the information is stored locally, which means that on average the database won't have to hold more than 1000 rows of information and all of the information will take on average half a second to load.

5. Findings and insights

Summary of major findings:

CPU Usage: Significant CPU spikes (up to 80-100%) were observed during database operations such as adding, editing, deleting, loading, and searching entries. Methods contributing to these spikes include those related to UI elements, database operations, and list handling.

Memory Management: No memory leaks were detected. Memory usage experienced temporary spikes during database operations but quickly returned to normal levels, indicating a negligible impact on overall performance.

Database scalability: Load testing with 1000 to 50000 insert/read operations showed that operation completion time increases proportionally with the data volume.

Performance bottlenecks:

High CPU Usage During Database Operations: The primary bottleneck appears to be the high CPU utilization during all tested database interactions (insert, edit, delete, load, search).

CPU Intensive UI Components/Methods: Methods such as TopBar, invoke, NavigationItem, SwipeToDelete, rememberSwipeBoxState, and ExercisesRow were identified as significant CPU consumers during related operations.

Potential Scalability Issues: The proportional increase in operation time with data volume suggests a potential bottleneck if the application were to scale to handle significantly larger datasets.

Areas for optimization:

Database Operation Optimization: Investigate the identified CPU-intensive database methods (upsert, update, getExercises) for potential optimizations, even if current performance is acceptable.

UI Rendering and List Handling: Optimize UI components and methods that contribute to CPU spikes during list loading and rendering.

Background Threads/Coroutines: Ensure that all computationally intensive database operations are consistently performed on background threads to prevent UI freezes and improve responsiveness.

Caching: For frequently accessed data that doesn't change often, implementing a caching strategy could reduce redundant database queries and CPU load.

6. Conclusion

The application demonstrates acceptable performance for its intended use case of managing workout information locally. Memory management is a strong point, with no leaks detected and efficient handling of temporary memory spikes. CPU usage is high during database-intensive tasks; however, this is partly attributed to the emulator environment and the nature of these operations. For the typical data load (under 1000 rows), operations are completed quickly.

Based on the findings, the application appears to meet expected performance thresholds for its current scope, where data is stored locally and database sizes are relatively small (e.g., loading information in approximately half a second). The manual testing also suggests that the user experience related to performance is likely satisfactory under normal conditions.

Next steps:

Real Device Testing: Prioritize testing on a range of physical devices to validate CPU usage patterns and confirm if the high percentages observed on the emulator translate to real-world performance issues.

Targeted CPU Optimization: If real-device testing confirms performance concerns, focus on optimizing the specific methods identified as high CPU consumers during database and UI rendering operations.

Implement Pagination (Future Scalability): If there's any plan to support significantly larger datasets in the future, proactively design and implement pagination for list loading.

Refine Background Processing: Double-check that all database operations are consistently offloaded from the main thread using Kotlin Coroutines or other multi-threading mechanisms to ensure UI responsiveness.

Continuous Monitoring: Continue to use profiling tools throughout the development lifecycle to catch any new performance regressions or memory issues as new features are added.