Gage University College

Computer Science Department

Date: 5/6/2025

Course: Android Application Development

Group Members:

Fuad Ahmed ID 023/14

Eyerus Daniel ID 018/14

Firehiwot Lakew ID 019/14

1. Introduction

What is Gage App?

The Gage App is a mobile application designed to help people track their fitness and wellness activities. It's like a digital notebook and timer combined, specifically for workouts and hydration. It's a personal fitness and wellness tracker. Its focus is on you and your personal health goals.

2. Objective of the project

Many people want to improve their health, but it's hard to remember to drink enough water or keep track of workout times. It's also hard to look back at your progress if you don't write anything down.

The Solution: The Gage App solves this problem by making it easy to log your exercise and water intake, and create an exercise list, all in one place.

Benefits:

Organization: It helps you stay organized with your workouts.

Tracking: It helps you track your progress.

Awareness: It helps you become more aware of your water intake.

Simplicity: It makes tracking these things simple and quick.

Date: It lets you see the current date.

Key Features

Workout Timer: An easy-to-use timer to track how long you've been exercising. You can start, pause, and reset it.

Water Counter: A simple counter to keep track of how much water you've consumed. You can increase the count each time you drink.

Exercise Log: A way to log the exercises you've done, how long you did them, and how many drinks you had while doing that exercise.

Current Date: The app displays the current date.

3. System Desing

Our Approach: Our Gage App uses a simplified version of MVVM.

MVVM helps us organize the app into distinct layers:

Model: The "data" part (the exercises, the water count, etc.).

View: The "what the user sees" part (the buttons, the timer display, the lists).

ViewModel: The "brain" that sits between the Model and the View. It prepares data for the View and handles user interactions.

The ViewModel is the heart of the app's logic. It's a layer that sits between the Model and the View.

In Our App:

MainViewModel: Our app has one main ViewModel, called MainViewModel. It does a lot of the heavy lifting.

Responsibility:

Manage State: The ViewModel holds the app's "state" (the current value of the timer, the list of exercises, etc.).

Handle Logic: It contains the logic for what happens when the user interacts with the View (e.g., when the user clicks "start," the ViewModel starts the timer).

Prepare Data: The ViewModel takes data from the Model (like the exercise list) and prepares it in a way that's ready for the View to display.

Update the View: When something changes in the ViewModel (like the timer ticking), it tells the View to update itself.

Hold Data: The MainViewModel holds all the data of the app. 3. The Flow: How It All Works Together Here's a breakdown of how the different layers interact:

Our Composables:

MainScreen: The main screen of the app, where everything is displayed.

CyclingTimer: The timer UI, including buttons to start, pause, and reset.

WaterCounter: The UI for counting drinks, with an "add" button and a reset button.

ExerciseList: The UI that shows the list of exercises and allows the user to add new ones.

Utils.kt: contains a function to help convert the seconds to minutes.

The Flow: How It All Works Together Here's a breakdown of how the different layers interact:

A. User Interaction (View)

The user clicks a button in the CyclingTimer to start the timer.

The CyclingTimer function tells MainViewModel that the user clicked "start."

B. Logic in ViewModel

MainViewModel has a function called startTimer() that's called when the user clicks "start."

startTimer() begins increasing the timeTracker variable every second.

startTimer() also updates a variable called timerRunning to true to tell the UI that the timer is now running.

MainViewModel also has a function for pause and reset.

C. Update the UI (View)

The MainViewModel exposes the timerTime as a StateFlow.

The CyclingTimer is "watching" this StateFlow and updates the timer time on the screen whenever it changes.

D. Add an Exercise

When the user added an exercise in ExerciseList component, it calls the addExercise() function in the MainViewModel.

The MainViewModel will create a new exercise and then reset the timer and the drinks count.

The MainViewModel will update the _exerciseAdded variable to update the UI.

E. Add Water

When the user clicks on the addWater button in the WaterCounter component, it calls the addWater() function in the MainViewModel.

The MainViewModel will update the _waterCount variable.

F. Display the current date

The MainViewModel will update the current date when the app starts and expose it with a StateFlow.

The MainScreen is watching the StateFlow and update the current date.

4. Tools and Technologies we used

1. Our Development Workshop or IDE

IDE (Integrated Development Environment) for Android app development.

Why We Use Android Studio:

Official: It's built and supported by Google, so it's perfectly suited for Android development.

Comprehensive: It has everything we need in one place:

Code Editor: A smart editor for writing Kotlin code.

UI Designer: Tools for creating the app's user interface (UI).

Debugger: Tools for finding and fixing problems in our code.

Build System: Tools for compiling and building the app.

Emulator: A virtual Android device on our computer, allowing us to test the app without a real phone.

2. Our Construction Language

Kotlin: is a modern programming language that is the official language for Android development.

Why We Use It:

Official Endorsement: Google has officially designated Kotlin as the preferred language for Android.

Safety: Kotlin helps us write safer code by reducing errors (like null pointer exceptions).

Conciseness: Kotlin lets us write the same code with fewer lines than Java, making it more readable.

Interoperability: Kotlin works seamlessly with existing Java code. If you have Java code, you can still use it with Kotlin.

Modern Features: Kotlin has modern language features that make coding more efficient and enjoyable, like extension functions and coroutines.

3. Our UI Construction Kit

Jetpack Compose: is a modern UI toolkit from Google, designed to make building user interfaces for Android apps easier and faster.

Why We Use It:

Declarative: Instead of telling the UI how to do something (like in the older XML-based system), we tell it what we want, and Compose figures out the "how."

Component-Based: We build the UI out of reusable pieces called "composables." Each composable is like a building block.

State-Driven: Compose automatically updates the UI when the app's state changes. This makes UI updates much simpler.

Kotlin-First: Compose is designed to work very well with Kotlin.

Live Previews: Android Studio provides live previews of our UI as we code it with Compose, speeding up development.

4. Our Design Blueprint

Material Design: is Google's latest design system. It's a set of guidelines and principles for creating modern and consistent user interfaces.

Why We Use It:
Consistency: It ensures that our app looks and feels like a modern Android app.
User Familiarity: Users are familiar with Material Design, so our app feels natural to use.
Accessibility: Material Design considers accessibility, making our app usable by more people.
Theme: Material Design allows us to easily set up our app's theme (colors, fonts, etc.).
Dynamic Colors: we are using this to make the app more personalized.

5. Our Project Manager and Builder

Gradle is a build automation system. It's like a project manager that helps us with Manage dependencies (external libraries), Compile our code, Build the app (create the APK file) and Run tests.

Why We Use It:

Standard: It's the standard build system for Android development.

Dependency Management: It makes it easy to add and manage external libraries (like the Jetpack libraries).

Customization: It's very flexible and can be customized for complex build processes.

5. Codes and Samples

```
package com.example.gage_app

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.viewModels
import androidx.compose.foundation.layout.fillMaxSize
```

```
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.ui.Modifier
import com.example.gage_app.GageAppTheme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val viewModel: MainViewModel by viewModels()
        setContent {
            GageAppTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    MainScreen(viewModel)
                }
            }
        }
    }
}
```

Purpose: This is the entry point of the app. It's where the app starts running.

Key Functionality:

onCreate(): This is called when the app starts.

setContent: This is a Jetpack Compose function that defines the UI of the activity.

GageAppTheme: call the theme of the app.

Surface: contains the MainScreen.

MainScreen(viewModel): This is where we call the MainScreen composable function and pass it the MainViewModel instance.

val viewModel: MainViewModel by viewModels(): get the viewmodel.

```
package com.example.gage_app
```

```
import androidx.compose.runtime.mutableStateListOf
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import java.text.SimpleDateFormat
import java.util.Calendar
import java.util.Locale

class MainViewModel : ViewModel() {
 // Cycling Timer State
    private val _timerTime = MutableStateFlow(0)
    val timerTime: StateFlow<Int> = _timerTime.asStateFlow()

    private val _timerRunning = MutableStateFlow(false)
    val timerRunning: StateFlow<Boolean> = _timerRunning.asStateFlow()

    // Water Counter State
    private val _waterCount = MutableStateFlow(0)
    val waterCount: StateFlow<Int> = _waterCount.asStateFlow()

    // Current Date
    private val _currentDate = MutableStateFlow("")
    val currentDate: StateFlow<String> = _currentDate.asStateFlow()

    //Exercise List
    private var _exerciseList = mutableStateListOf<Exercise>()
    val exerciseList: List<Exercise> = _exerciseList

    //new exercise
    private val _newExercise = MutableStateFlow("")
    val newExercise : StateFlow<String> = _newExercise.asStateFlow()

    // added exercise
    private val _exerciseAdded = MutableStateFlow(false)
    val exerciseAdded: StateFlow<Boolean> = _exerciseAdded.asStateFlow()

    //Current exercise index
    private var currentExerciseIndex = -1
    //time tracker
    private var timeTracker = 0

    init {
        updateCurrentDate()
    }
```

```kotlin
// Timer Functions
fun setTimerTime(time: Int) {
    _timerTime.value = time
}

fun startTimer() {
    _timerRunning.value = true
    viewModelScope.launch {
        while (_timerRunning.value) {
            increaseTimer()
            kotlinx.coroutines.delay(1000L)
        }
    }
}

fun pauseTimer() {
    _timerRunning.value = false
}

fun increaseTimer() {
    _timerTime.value += 1
    timeTracker++
}

fun resetTimer() {
    _timerTime.value = 0
    _timerRunning.value = false
    timeTracker = 0
}

// Water Counter Functions
fun addWater() {
    _waterCount.value += 1
}

fun resetWater() {
    _waterCount.value = 0
}

// Date function
private fun updateCurrentDate() {
    val calendar = Calendar.getInstance()
    val dateFormat = SimpleDateFormat("dd MMMM yyyy", Locale.getDefault())
    _currentDate.value = dateFormat.format(calendar.time)
}
```

```
    // Exercise list function
    fun addExercise(exerciseName: String) {
        val newExercise = Exercise(
            name = exerciseName,
            workoutTime = timeTracker,
            drinkCount = _waterCount.value // Capture the current number of drinks
        )
        _exerciseList.add(newExercise)
        currentExerciseIndex = _exerciseList.lastIndex
        resetTimer()
        resetWater() // reset the water count
        _exerciseAdded.value = !_exerciseAdded.value //Update UI
    }
    //new exercise
    fun onNewExerciseChange(newExercise: String){
        _newExercise.value = newExercise
    }
}
```

Purpose: The brain of the app. It manages the state and logic.

Key Functionality:

timerTime, timerRunning, waterCount, currentDate: These are StateFlow objects that hold the state of the timer, water count, date and etc.

exerciseList: contains the list of the exercise.

startTimer(), pauseTimer(), resetTimer(): Timer logic.

addWater(), resetWater(): Water counter logic.

addExercise(): Adds an exercise to the list.

updateCurrentDate(): update the current date.

```
package com.example.gage_app

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
```

```
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp

@Composable
fun MainScreen(viewModel: MainViewModel) {
    val timerTime by viewModel.timerTime.collectAsState()
    val timerRunning by viewModel.timerRunning.collectAsState()
    val waterCount by viewModel.waterCount.collectAsState()
    val currentDate by viewModel.currentDate.collectAsState()
    val exercises = viewModel.exerciseList
    val newExercise by viewModel.newExercise.collectAsState()
    val exerciseAdded by viewModel.exerciseAdded.collectAsState()

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(text = "Today is: $currentDate")
        CyclingTimer(
            timerTime = timerTime,
            isRunning = timerRunning,
            startTimer = { viewModel.startTimer() },
            pauseTimer = { viewModel.pauseTimer() },
            resetTimer = { viewModel.resetTimer() }
        )
        WaterCounter(
            count = waterCount,
            onAddWater = { viewModel.addWater() },
            onResetWater = { viewModel.resetWater() }
        )
        ExerciseList(
            exercises = exercises,
            addExercise = { viewModel.addExercise(it) },
            newExercise = newExercise,
            onNewExerciseChange = {viewModel.onNewExerciseChange(it)},
            exerciseAdded = exerciseAdded
        )
    }
}
```

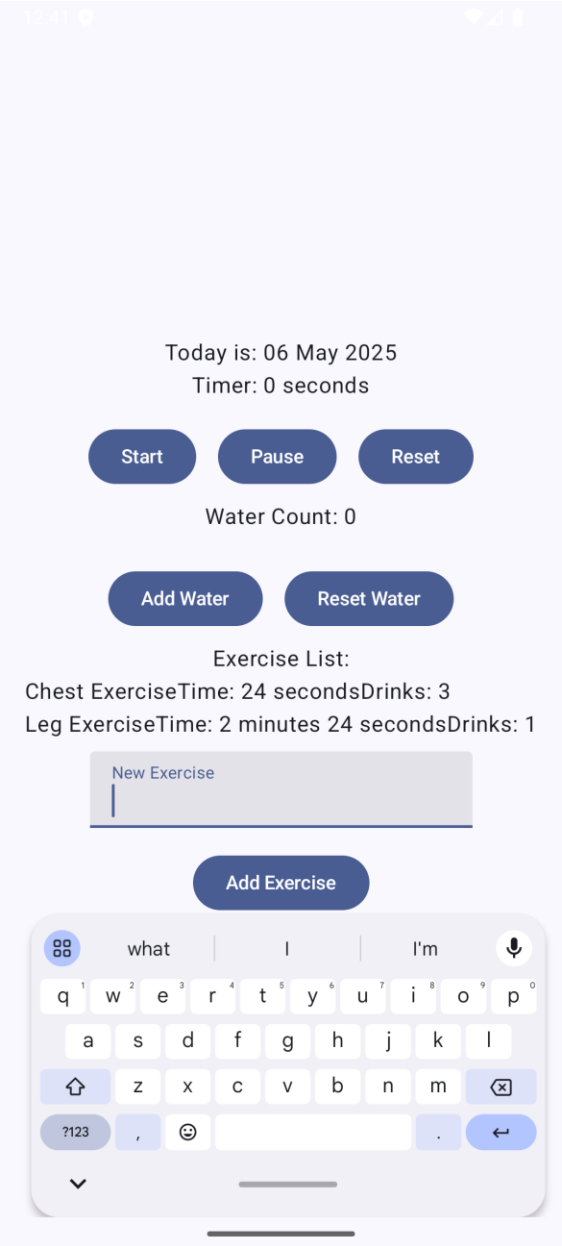
Purpose: The main UI screen that combines all the other UI elements.

Key Functionality:

CyclingTimer: Displays the timer.

WaterCounter: Displays the

5. Screen Shots of the app



6. Conclusion

The Gage App - A Simple, Modern Fitness Tracker The Gage App is a mobile application designed to help users track their workout time, monitor their water intake, and log their exercise routines. It is a simple and easy-to-use phone app that helps you keep track of your workouts and water drinking. It looks modern and was built using the best tools available for Android apps. It does what it's supposed to do well, and it's set up so that we could easily add more features to it in the future.