UNIVERSIDAD TECNOLÓGICA DEL CENTRO DE VERACRUZ

Reporte de actividad realizada en clase

Nombre: Alvizar Martínez Alexis

Grupo: 9 “B” IDGS

EDUCACIÓN SECRETARÍA DE EDUCACIÓN PÚBLICA
UTP COORDINACIÓN GENERAL DE UNIVERSIDADES TECNOLÓGICAS Y POLITÉCNICAS
VERACRUZ GOBIERNO DEL ESTADO
SEV Secretaría de Educación
DET Dirección de Educación Tecnológica del Estado de Veracruz
SEMSyS Subsecretaría de Educación Media Superior y Superior

1. Propósito General

Este documento detalla el desarrollo de una aplicación móvil orientada a dispositivos Wear OS, la cual integra en una sola interfaz las funciones de cronómetro y temporizador. La propuesta se enfoca en lograr una experiencia de usuario sencilla y eficiente, adecuando el diseño a las características de pantallas circulares, propias de los relojes inteligentes.

---

2. Características y Operaciones Principales

El sistema cuenta con dos funciones clave, accesibles mediante una interfaz práctica y minimalista:

- **Modo Cronómetro:** Permite iniciar un conteo desde cero, con opciones para detenerlo, pausarlo o reiniciarlo, según las necesidades del usuario.
- **Modo Temporizador:** Activa una cuenta regresiva al seleccionar un tiempo definido (15, 20 o 30 minutos), concluyendo automáticamente cuando el tiempo finaliza.
- **Controles de Interfaz:**
    - *Botón de Inicio/Pausa:* Cambia entre estados activos e inactivos del cronómetro o temporizador.
    - *Botón de Reinicio:* Cancela el conteo actual y restablece el modo en ejecución.
- **Selección Automática de Modo:** El temporizador se activa tras elegir un tiempo específico; si no se selecciona ninguno, se asume el modo cronómetro.
- **Diseño Optimizado:** La interfaz se implementó con Jetpack Compose y elementos visuales adaptados para dispositivos con Wear OS, priorizando legibilidad y navegación simple.

---

3. Ilustracion del Funcionamiento

---

4. Descripción Técnica del Código (Resumen Funcional)

La lógica de la aplicación fue desarrollada en Kotlin usando componentes de Jetpack Compose. Se emplearon efectos asíncronos para gestionar la actualización constante del tiempo. Entre los componentes implementados:

- Control del estado de ejecución mediante variables reactivas.
- Cálculo y formato del tiempo en minutos, segundos y centésimas.
- Inclusión de elementos como Chip, Button y Text para interacción dinámica.
- Interfaz completamente responsive, diseñada para dispositivos circulares.

La estructura del código está contenida dentro de MainActivity, segmentada en funciones composables que permiten la modularidad y fácil prueba en emuladores de Wear OS.

Código

```
package com.example.cronometro.presentation
import android.os.Bundle
import android.os.SystemClock
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.wear.compose.material.*
import androidx.wear.tooling.preview.devices.WearDevices
import com.example.cronometro.presentation.theme.CronometroTheme
import kotlinx.coroutines.delay
```

```
class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContent {
            CronometroTheme  {
                Box(
                    modifier = Modifier
                        .fillMaxSize()
                        .background(MaterialTheme.colors.background)
                ) {
                    StopwatchTimerScreen(modifier = Modifier.fillMaxSize())

                    TimeText(
                        modifier = Modifier
                            .align(Alignment.TopCenter)
                            .padding(top = 4.dp)
                    )
                }
            }
        }
    }

    @Composable
    fun StopwatchTimerScreen(modifier: Modifier = Modifier) {
        var isRunning by remember { mutableStateOf(false) }
        var isTimerMode by remember { mutableStateOf(false) }
        var selectedTime by remember { mutableStateOf(0L) }
```

```
    var elapsed by remember { mutableStateOf(0L) }
    var baseTime by remember { mutableStateOf(0L) }

    LaunchedEffect(isRunning) {
        if (isRunning) {
            baseTime = SystemClock.elapsedRealtime() - elapsed
            while (isRunning) {
                elapsed = SystemClock.elapsedRealtime() - baseTime
                if (isTimerMode && elapsed >= selectedTime) {
                    isRunning = false
                    break
                }
                delay(50L)
            }
        }
    }

    val displayTime = if (isTimerMode) {
        val remaining = (selectedTime - elapsed).coerceAtLeast(0)
        remaining
    } else {
        elapsed
    }

    val minutes = (displayTime / 60_000).toInt()
    val seconds = ((displayTime % 60_000) / 1_000).toInt()
    val centis = ((displayTime % 1_000) / 10).toInt()

    val timeString = String.format("%02d:%02d:%02d", minutes, seconds, centis)
```

```
    Column(
        modifier = modifier.padding(8.dp),
        horizontalAlignment  = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        Row(
            modifier = Modifier.padding(top = 24.dp),
            horizontalArrangement  = Arrangement.spacedBy(8.dp)
        ) {
            listOf(15, 20, 30).forEach { min ->
                Chip(
                    onClick = {
                        selectedTime = min * 60_000L
                        isTimerMode = true
                        elapsed = 0L
                        isRunning = false
                    },
                    label = {
                        Text(text = "$min", fontSize = 12.sp)
                    },
                    modifier = Modifier.size(48.dp)
                )
            }
        }

        Text(
            text = timeString,
            fontSize = 36.sp
        )
```

```
        Row(horizontalArrangement  = Arrangement.spacedBy(12.dp)) {
            Button(
                onClick = {
                    isRunning = !isRunning
                }
            ) {
                Text(
                    text = if (isRunning) "Detener" else "Iniciar",
                    fontSize = 12.sp
                )
            }
            Button(
                onClick = {
                    isRunning = false
                    elapsed = 0
                    selectedTime = 0
                    isTimerMode = false
                }
            ) {
                Text(
                    text = "Reiniciar",
                    fontSize = 12.sp
                )
            }
        }
    }
}

@Preview(
    showSystemUi = true,
```

```
    showBackground = true,
    device = WearDevices.SMALL_ROUND
)
@Composable
fun Preview_StopwatchScreen() {
    CronometroTheme {
        Box(
            modifier = Modifier
                .fillMaxSize()
                .background(MaterialTheme.colors.background)
        ) {
            StopwatchTimerScreen(modifier = Modifier.fillMaxSize())

            TimeText(
                modifier = Modifier
                    .align(Alignment.TopCenter)
                    .padding(top = 4.dp)
            )
        }
    }
}
```

---

5. Evaluación Final

Se completó con éxito la creación de una herramienta dual de medición temporal para Wear OS, destacando por su simplicidad visual y eficacia funcional. La solución permite a los usuarios gestionar intervalos de tiempo de forma cómoda en entornos móviles compactos, sin comprometer el control ni la visibilidad de la información presentada.