```swift
//
//  Created by Egzon PLLANA on 26.7.24.
//

import UIKit
import SwiftUI

/// Converts degrees to radians.
private extension CGFloat {
    var radians: CGFloat {
        return self * .pi / 180.0
    }
}

/// Represents different types of layers in the gauge view.
private enum LayerType {
    case gauge
    case background
}

/// Custom clear color used in the gradient.
private extension UIColor {
    static let customClear = UIColor(white: 1.0, alpha: 0.0)
}

/// Constants used for drawing and configuring the gauge.
private enum Constants {
    static let padding12: CGFloat = 12.0
    static let padding16: CGFloat = 16.0
    static let padding40: CGFloat = 40.0
    static let animationTime: Double = 10.0
}

/// A UIView subclass that displays a gauge with customizable appearance and animated value indicator.
class GaugeViewXK: UIView {

    // MARK: - Properties -
    private let gaugeValues: GaugeValues
    private let gaugeColor: GaugeColor
    private let gaugeWidth: CGFloat
    private let gaugeBackgroundColor: UIColor
    private let indicatorColor: UIColor
    private let indicatorWidth: CGFloat
    private let labelColor: UIColor
    private let labelFont: UIFont

    private var currentValue: CGFloat = 0.0
    private var targetValue: CGFloat = 0.0
    private var animationTime: Double
    private var displayLink: CADisplayLink?
    private let valueLabel = UILabel()

    private let startAngle: CGFloat = CGFloat(120).radians
    private let endAngle: CGFloat = CGFloat(60).radians
    private let totalAngle: CGFloat = CGFloat(300).radians
    private var indicatorLayer: CAShapeLayer?

    // MARK: - Initializers -
    /// Initializes a new GaugeViewXK with customizable properties.
    init(
        gaugeValues: GaugeValues = .range(start: 10, end: 100, parts: 10),
        gaugeColor: GaugeColor = .gradient([.red, .yellow, .green]),
        gaugeBackgroundColor: UIColor = .clear,
        gaugeWidth: CGFloat = 18,
        indicatorColor: UIColor = UIColor(Color.primary),
        indicatorWidth: CGFloat = 4,
        labelColor: UIColor = UIColor(Color.primary),
        labelFont: UIFont = .systemFont(ofSize: 12),
        animationTime: Double = Constants.animationTime
    ) {
        self.gaugeValues = gaugeValues
        self.gaugeColor = gaugeColor
        self.gaugeWidth = gaugeWidth
        self.gaugeBackgroundColor = gaugeBackgroundColor
        self.labelColor = labelColor
        self.labelFont = labelFont
        self.indicatorColor = indicatorColor
        self.indicatorWidth = indicatorWidth
        self.animationTime = animationTime

        super.init(frame: .zero)
        setupDisplayLink()
        setupValueLabel()
    }

    /// Required initializer for using GaugeViewXK with storyboards.
    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    // MARK: - Drawing Methods -
    override func draw(_ rect: CGRect) {
        super.draw(rect)
        drawGauge(layerType: .background)
        drawGauge(layerType: .gauge)
    }

    override func layoutSubviews() {
        super.layoutSubviews()
        drawLabels(in: bounds)
        drawIndicator(in: bounds)
    }

    /// Sets the target value of the gauge and initiates the animation.
    /// - Parameter value: The new value to set.
    func setValue(_ value: Double) {
        let minValue: CGFloat
        let maxValue: CGFloat
```

```swift
        // Make sure the given value is valid.
        switch gaugeValues {
        case .range(let start, let end, _):
            minValue = start
            maxValue = end
        case .values(let array):
            guard array.count > 1,
                  let firstValue = array.first,
                  let lastValue = array.last
            else {
                minValue = 0
                maxValue = 0
                return
            }
            minValue = firstValue
            maxValue = lastValue
        }
        guard CGFloat(value) >= minValue && CGFloat(value) <= maxValue else
         {
            return
        }
        self.targetValue = CGFloat(value)
        if displayLink == nil {
            setupDisplayLink()
        }
    }
}

// MARK: - Private Extension for Gauge Drawing -
private extension GaugeViewXK {
    /// Draws the gauge based on the specified layer type.
    func drawGauge(layerType: LayerType) {
        let rect = self.bounds
        let center = CGPoint(x: rect.midX, y: rect.midY)
        let radius = rect.width / 2 - Constants.padding12
        let path = createArcPath(center: center, radius: radius, layerType:
         layerType)
        let shapeLayer = createShapeLayer(path: path.cgPath)

        switch layerType {
        case .gauge:
            configureGaugeLayer(shapeLayer: shapeLayer, rect: rect)
        case .background:
            shapeLayer.strokeColor = gaugeBackgroundColor.cgColor
            layer.addSublayer(shapeLayer)
        }
    }

    /// Creates an arc path for the gauge.
    func createArcPath(center: CGPoint, radius: CGFloat, layerType:
     LayerType) -> UIBezierPath {
        return UIBezierPath(
            arcCenter: center,
            radius: radius,
            startAngle: startAngle,
            endAngle: endAngle,
            clockwise: true
        )
    }

    /// Creates a shape layer with the specified path.
    func createShapeLayer(path: CGPath) -> CAShapeLayer {
        let shapeLayer = CAShapeLayer()
        shapeLayer.path = path
        shapeLayer.lineWidth = gaugeWidth
        shapeLayer.fillColor = UIColor.clear.cgColor
        return shapeLayer
    }

    /// Configures the appearance of the gauge layer based on the specified color.
    func configureGaugeLayer(shapeLayer: CAShapeLayer, rect: CGRect) {
        shapeLayer.strokeColor = UIColor.black.cgColor

        switch gaugeColor {
        case .single(let color):
            shapeLayer.strokeColor = color.cgColor
            layer.addSublayer(shapeLayer)
        case .gradient(let colors):
            applyGradientLayer(colors: colors, shapeLayer: shapeLayer,
             rect: rect)
        }
    }

    /// Applies a gradient layer to the gauge.
    func applyGradientLayer(colors: [UIColor], shapeLayer: CAShapeLayer,
     rect: CGRect) {
        let gradientLayer = CAGradientLayer()
        gradientLayer.type = .conic
        gradientLayer.startPoint = CGPoint(x: 0.5, y: 0.5)
        gradientLayer.endPoint = CGPoint(x: 0.23, y: 2)
        gradientLayer.locations = calculateGradientLocations(for: colors)
        gradientLayer.frame = rect
        gradientLayer.colors = processGradientColors(colors: colors).map {
         $0.cgColor }
        gradientLayer.mask = shapeLayer
        layer.addSublayer(gradientLayer)
    }

    /// Processes the gradient colors to adjust for clear color.
    func processGradientColors(colors: [UIColor]) -> [UIColor] {
        var adjustedColors = colors
        if colors.count > 1, let lastColor = colors.last, lastColor ==
         .clear {
            adjustedColors.removeLast()
            let clearColor = UIColor(white: 1.0, alpha: 0.0)
            adjustedColors.append(clearColor)
        }
        return adjustedColors
    }
```

```swift
    /// Calculates the gradient locations for the specified colors.
    func calculateGradientLocations(for colors: [UIColor]) -> [NSNumber] {
        guard colors.count > 1 else { return [0.0, 1.0] }

        let locationIncrement = 0.8 / Double(colors.count - 1)
        var locations: [NSNumber] = (0..<(colors.count - 1)).map {
         NSNumber(value: Double($0) * locationIncrement) }

        if let lastColor = colors.last, lastColor == .clear {
            locations.append(0.9)
        }
        locations.append(1.0)

        return locations
    }
}

// MARK: - Private Extension for Label Drawing -
private extension GaugeViewXK {
    /// Draws labels around the gauge.
    func drawLabels(in rect: CGRect) {
        // Remove existing labels except for valueLabel
        subviews.filter { $0 !== valueLabel }.forEach {
         $0.removeFromSuperview() }

        let center = CGPoint(x: rect.midX, y: rect.midY)
        let gaugeRadius = rect.width / 2 - Constants.padding12
        let labelRadius = gaugeRadius - gaugeWidth / 2 - Constants.padding16
        let labelValues = getLabelValues()

        labelValues.enumerated().forEach {
            index,
            value in
            let angle = calculateLabelAngle(for: index, total:
             labelValues.count)
            let position = calculateLabelPosition(
                center: center,
                radius: labelRadius,
                angle: angle
            )

            let label = createLabel(with: value)
            label.center = position

            addSubview(label)
        }
    }

    /// Retrieves the label values from the gauge values.
    func getLabelValues() -> [CGFloat] {
        switch gaugeValues {
        case .range(let start, let end, let parts):
            return stride(from: start, through: end, by: (end - start) /
             CGFloat(parts)).map { $0 }
        case .values(let values):
            return values
        }
    }

    /// Calculates the angle for a label at the specified index.
    func calculateLabelAngle(for index: Int, total: Int) -> CGFloat {
        return startAngle + (CGFloat(index) / CGFloat(total - 1)) *
         totalAngle
    }

    /// Calculates the position for a label based on the angle and radius.
    func calculateLabelPosition(center: CGPoint, radius: CGFloat, angle:
     CGFloat) -> CGPoint {
        let x = center.x + radius * cos(angle)
        let y = center.y + radius * sin(angle)
        return CGPoint(x: x, y: y)
    }

    /// Creates a UILabel for the specified value.
    func createLabel(with value: CGFloat) -> UILabel {
        let label = UILabel()
        label.text = "\(Int(value))"
        label.font = labelFont
        label.textColor = labelColor
        label.sizeToFit()
        label.frame.origin = CGPoint(
            x: label.frame.origin.x - label.bounds.width / 2,
            y: label.frame.origin.y - label.bounds.height / 2
        )
        return label
    }

    /// Sets up the value label.
    func setupValueLabel() {
        valueLabel.textColor = labelColor
        valueLabel.font = labelFont
        addSubview(valueLabel)
    }
}

// MARK: - Private Extension for Indicator Drawing -
private extension GaugeViewXK {

    /// Draws the indicator if not already drawn.
    func drawIndicator(in rect: CGRect) {
        if indicatorLayer == nil {
            createIndicatorLayer()
        }
        updateIndicatorPath()
    }

    /// Creates and configures the indicator layer.
    func createIndicatorLayer() {
        let newIndicatorLayer = CAShapeLayer()
        newIndicatorLayer.strokeColor = indicatorColor.cgColor
        newIndicatorLayer.lineWidth = indicatorWidth
        newIndicatorLayer.lineCap = .round
        layer.addSublayer(newIndicatorLayer)
        indicatorLayer = newIndicatorLayer
    }

    /// Updates the path of the indicator layer.
    func updateIndicatorPath() {
        guard let indicatorLayer = indicatorLayer else { return }

        let center = CGPoint(x: bounds.midX, y: bounds.midY)
        let radius = bounds.width / 2 - gaugeWidth - Constants.padding40

        let endAngle = calculateEndAngle()
        let endPoint = CGPoint(
            x: center.x + radius * cos(endAngle - CGFloat(90).radians),
            y: center.y + radius * sin(endAngle - CGFloat(90).radians)
        )

        let indicatorPath = UIBezierPath()
        indicatorPath.move(to: center)
        indicatorPath.addLine(to: endPoint)

        indicatorLayer.path = indicatorPath.cgPath
        indicatorLayer.lineWidth = indicatorWidth
    }

    /// Calculates the end angle for the indicator based on the current value.
    func calculateEndAngle() -> CGFloat {
        let degrees = mapValueToDegrees(value: currentValue)
        let scaleFactor: CGFloat = 300 / 360
        let baseAngle: CGFloat = 210
        return (baseAngle + degrees * scaleFactor).radians
    }

    /// Sets up a display link to animate the indicator.
    func setupDisplayLink() {
        displayLink = CADisplayLink(target: self, selector:
         #selector(updateNextValue))
        displayLink?.add(to: .main, forMode: .default)
    }

    /// Updates the current value and indicator path.
    @objc func updateNextValue() {
        let valueDifference = targetValue - currentValue

        if abs(valueDifference) > 0.1 {
            let changeRate = valueDifference / CGFloat(animationTime)
            currentValue = min(
                max(
                    currentValue + changeRate,
                    min(targetValue, currentValue)
                ),
                max(targetValue, currentValue)
            )
            updateIndicatorPath()
            valueLabel.text = "\(Int(currentValue))"
        } else {
            finalizeUpdate()
        }
    }

    /// Finalizes the update when the target value is reached.
    func finalizeUpdate() {
        currentValue = targetValue
        updateIndicatorPath()
        valueLabel.text = "\(Int(currentValue))"
        displayLink?.invalidate()
        displayLink = nil
    }

    /// Maps a value to its corresponding angle in degrees.
    func mapValueToDegrees(value: CGFloat) -> CGFloat {
        guard let range = gaugeValues.minMax() else {
            return 0.0
        }
        let normalizedValue = (value - range.min) / (range.max - range.min)
        return normalizedValue * 360.0
    }
}
```