# Take-Home Assignment: Rate Limiting Module for go-service-kit

## Context

go-service-kit is a shared internal/external toolkit for building Go microservices. Anything added must be:

- Safe, well-tested, and production-oriented
- Ergonomic and future-proof (avoid locking us into premature design)
- Concurrency-safe and allocation-conscious
- Observable (metrics / tracing seams)
- Easy for many downstream teams to adopt incrementally

You will design and implement an initial ratelimit module. Assume this will be widely used across HTTP handlers, background workers, and potentially future distributed setups.

You do NOT need to integrate with any real metrics library—just provide integration seams.

## Problem Statement

Implement a pluggable, per-key, in-memory rate limiting package with:

1. A minimal core interface:
   type Limiter interface {
   Allow(ctx context.Context, key string) bool
   Wait(ctx context.Context, key string) error // blocks until capacity or ctx done
   }

2. At least one fully working Rate Limiting Algorithms.
3. Per-key limiting with bounded memory: configurable max tracked keys + eviction policy (LRU or TTL or hybrid). Document your choice & trade-offs.
4. Concurrency safe.
5. Structured error for rate limit denials.

## Non-Goals (You MAY mention how you would extend later)

- No need to implement a distributed store (e.g., Redis)
- No need for real HTTP integration code

## Deliverables

Provide a go module that any developer could use to implement a rate limiter and any additional methods that you see fit