Задание для создания микросервиса:

В этом задании используя инструменты параллельных вычислений, вы добавите сервису фоновый режим исполнения и повторяющиеся задачи.

Чтобы решить предыдущее задание на создание сервиса, вам пришлось в методе Check структуры Checker реализовать цикл, который будет проверять все элементы. Но так мы не используем все возможности go и выполняем проверки последовательно. Добавив параллельность мы можем ускорить проверки и выполнять несколько задач одновременно. Для этого мы могли бы просто каждую проверку элемента запускать в отдельной горутине. Но это будет не совсем удобно. В этом случае у нас проверка произойдет один раз и больше не повторится.

Давайте реализуем в чекере метод Run(), который запускает проверки в фоновом режиме и каждые 5 секунд пробегается по всем элементам чекера, осуществляя проверку. Он запускает проверки и завершает свою работу.

После того как мы справились с запуском проверок в фоне, нам нужна возможность добавлять элементы в уже запущенные проверки и эти элементы также должны начать проверяться. У нас уже есть метод Add, но давайте его улучшим. Если вы обратите внимание, то теперь наши проверки выполняются в фоновом режиме и метод Add() может привести к состоянию гонки (race condition), так как слайс items может параллельно изменяться из разных горутин. Этот нюанс необходимо доработать. (подсказка - используйте канал)

И нам остается реализовать возможность остановить наши запущенные проверки при желании. Для этого в структуру Checker нужно добавить новый метод Stop(). При вызове метода “Stop” горутина, которая в фоне выполняет проверки, должна

остановиться (завершить выполнение) и вывести на экран “Проверки остановлены”

(подсказка - используйте контекст)

Мы научили наш сервис стартовать в фоновом режиме и останавливаться по требованию. Каждые проверки запускаются в разных горутинах и мы общаемся между ними по каналам. Нам осталось учесть один момент, что если в горутине произойдет паника? Тогда наш сервис упадет. Давайте исправим это. Нужно написать код, который позволит при каждом запуске проверок восстанавливаться после паник. И если случилась паника, мы должны создать ошибку и вывести ее на экран.

*Как вы могли заметить, когда мы создавали структуру GoMetrClient, то объявляли ей пол timeOut целочисленным значением и условились, что это значение в секундах.

Давайте используем это. В методе Health этой структуры мы вызываем метод getHealth(), который возвращает нам состояние здоровья сервиса. Нужно в методе Health теперь осуществлять вызов getHealth с учетом таймаута у структуры. (Подсказка: для этого можете воспользоваться таймером, о котором мы говорили. Можно создать таймер тем путем, который был показан на лекции, или самостоятельно познакомиться с методом time.After() и воспользоваться им. По сути это тоже самое, что создание нового таймера, но в более удобном варианте для нашей конкретной задачи)