Задание для создания микросервиса:

Вы написали приложение, которое умеет вызывать сервис по расписанию и забирать оттуда метрики. Пришло время привести его в Production-ready вид и организовать правильную структуру и хранение в базе.

Последний раз вы реализовали проверки с помощью горутин и слайса items, висящего в памяти. Теперь наша задача - получать и хранить метрики, и после выключения приложения, что потребует использования базы данных.

Следующей задачей сервиса будет HTTP API для отдачи данных, структурированных по времени и возможностью фильтрации.

Сервис gometr предоставляет метрики по запросу адрес_сервера:порт/metrics в формате prometheus.

Формат выглядит следующим образом:

```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
go_gc_duration_seconds_sum 0
go_gc_duration_seconds_count 0
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 8
```

… и так далее

Строки начинающиеся с # представляют собой комментарии, которые хранить нет необходимости остальные параметры представляют собой строки вида **“имя значение”,** разделенные пробелом. В именах параметров можно встретить

дополнительные атрибуты в фигурных скобках, но для этого задания мы будем их считать частью имени самого параметра.

Вводная информация:

База данных - в корне вашего шаблона будет лежать docker-compose файл, содержащий в себе СУБД Postgres. Вам понадобится docker вместе с сервисом compose, который можно скачать с официального сайта docker. Я рекомендую сборку docker for desktop, которая позволяет сразу видеть все запущенные контейнеры и удобно с ними работать.

Основные действия:

Вначале организуем структуру команд для нашего сервиса, как было рассказано в лекции - создадим папки api и internal, папку cmd для хранения команд и отрефакторим наш сервис для запуска внутри этого сервиса.

Структура моделей

Для хранения данных в нашей базе, создадим структуру и положим в папку internal/models. Структура должна хранить в себе Name в строковом виде для имени метрики и Value для ее значения (так же в строковом виде). Помимо этого мы будем хранить Date - для обозначения времени забора метрики.

Для создания структуры базы данных - необходима всего одна таблица с полями id, timestamp, name и value, а также *индексы.*

Индексы - это специальные конструкции реляционных СУБД, упрощающие поиск по полям. В нашем случае, индексы должны быть расставлены по timestamp и name.

Подробнее про индексы.

Как создавать таблицы и поля в БД: https://metanit.com/sql/postgresql/2.2.php.

Нам понадобятся типы данных INTEGER для id, TEXT для name и value и timestamp для timestamp. Поскольку id является первичным ключом, он должен иметь атрибут serial, для автоматической вставки с увеличением значений (автоинкремента).

Как добавлять индексы в БД: https://tproger.ru/articles/indeksy-v-postgresql/

Рекомендую все запросы положить в один файл, в папку migrations в проекте - это упростит пересборку базы, а также жизнь остальным пользователям. Запросы для создания структуры таблиц, должны лежать в файлах с расширением .sql .

Работа с базой данных и запросами

Для работы с БД и описания запросов положим наш Storage класс в папку internal/db и напишем методы Add, List с фильтрами по дате и имени.

Для метода Add вероятнее всего придется сделать транзакцию, поскольку вам понадобится вставлять несколько столбцов сразу и лучше это сделать надежно, чтобы не было ситуации когда часть метрик пропала.

Для поиска по времени пригодятся операторы >= и <= .

Больше информации про транзакции - https://postgrespro.ru/docs/postgrespro/10/tutorial-transactions

Сбор данных

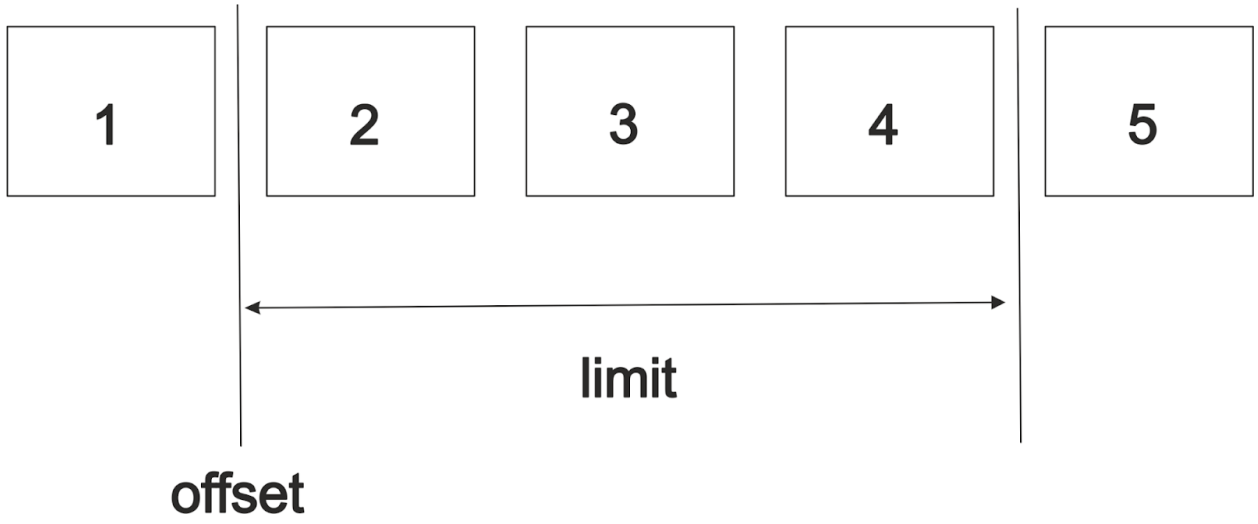
Для сбора данных с сервиса gometr используйте MetricsProcessor с методом add, реализующим логику забора метрик в формате Prometheus. Необходимо написать логику для снятия метрик, где мы будем опрашивать URL metrics нашим клиентом и парсить строку за строкой пропуская комментарии. Остальные строки достаточно бить по пробелу и сохранять, используя первый параметр как name а второй как value. Финальное сохранение данных производить в БД, используя наш Storage.

Время связывать логику воедино

Используйте main в корне internal и создайте в нем структуру App с методами Start() и Shutdown(). Начнем с добавления запуска с периодическими интервалами. Для установки интервалов вместо горутин используйте go cron из модуля https://github.com/go-co-op/gocron. Добавьте его в mod и вынесите scheduling в main.

Метод для сбора информации из сервиса

Последнее что нам необходимо сделать - создать метод для забирания информации из нашего сервиса в виде отсортированного по времени списка метрик. Метод должен содержать в себе пагинацию в виде limit (количество записей, которое мы берем из базы) и offset (с какой записи мы берем это из базы.



В запросе укажите эти параметры как LIMIT и OFFSET.

Также вам понадобятся два фильтра time_from (**с** какого временного промежутка собираются данные) и time_to - (**до** какого промежутка). Я предлагаю использовать формат времени web.

Например: 1971-01-02T00:00Z - второе января 1971 года полночь.

Также понадобится фильтр по имени метрики, аналогичный фильтрам из лекции. Name=go должен показать только метрики содержащие go в имени.

Например: go_gc_duration_seconds

Эти фильтры являются опциональными и все применяются в GET методе /metrics/list.

Возвращаемые значения сервиса должны быть обернуты в JSON.

На выходе вы получите сервис, который собирает, хранит и возвращает значения в привычном пользователям формате.