Тестовое задание

Язык: golang

Цель:

Задание составное и состоит из пунктов. Цель состоит в том, чтобы за выделенный срок выполнить как можно больше пунктов.

Задание:

Написать REST API сервис с несколькими доступными командами

1. Путь **/rest/substr**

   Нужно найти максимальную подстроку, не содержащую повторяющихся символов. Входная строка состоит из латинских букв (нижний и верхний регистр) и цифр.
   Доступные endpoint'ы:
   1.1 **POST /rest/substr/find —** endpoint для нахождения заданной подстроки. Строка находится в теле HTTP запроса.

   Для всех команд необходимо написать юнит тесты в отдельном файле.

2. Путь **/rest/email**

   Необходимо реализовать поиск строки следующего формата:
   «Email:__$email»
   Где вместо «__» может быть любое количество пробельных символов (в том числе и переносов строк), а вместо $email должна быть строка, похожая на настоящий email

   Доступные endpoint'ы:
   2.1 **POST /rest/email/check** — endpoint, которая анализирует тело HTTP запроса и выдает все найденные email адреса.

   **** Усложненное задание: написать аналогичный функционал, который вместо email адресов будет искать последовательность цифр, которая является корректным ИИН.***

3. Путь **/rest/counter**.

   Нужно написать простую реализацию счетчика. Необходимо добавить команды по увеличению и уменьшению счетчика. Хранить счетчик в Redis.
   Доступные endpoint'ы:
   *3.1 **POST /rest/counter/add/$i —** команда для увеличения счетчика, где **$i** — это целое значение, на которое следует увеличить счетчик*
   *3.2 **POST /rest/counter/sub/$i —** команда для уменьшения счетчика, где **$i** — это целое значение, на которое следует уменьшить счетчик*
   *3.3 **GET /rest/counter/val —** команда для получения текущего значения счетчика*

Для всех команд необходимо написать юнит тесты в отдельном файле.

**** Усложненный вариант задания. Написать юнит тесты без поднятия Redis сервера.***

4. Путь **/rest/user**

   Нужно написать эндпоинты для реализации CRUD (Create-Read-Update-Delete) операций над юзером. У юзера есть его ID (генерируется во время Create), имя, фамилия. Юзеры должны хранится в реляционной базе данных (SQL).
   Доступные endpoint'ы:
   4.1 **POST /rest/user —** endpoint для создания юзера. В HTTP теле запроса ожидаются следующие параметры:
   * first_name — имя пользователя
   * last_name — фамилия
   Сервер должен добавить в базу данных нового пользователя и вернуть ID, присвоенный базой в HTTP теле ответа.
   4.2 **GET /rest/user/:id** — endpoint для получения информации о пользователе (имя, фамилия) по id. ID передается как часть URL (вместо строки «:id»).
   4.3 **PUT /rest/user/:id** - endpoint для обновления информации о юзере. ID передается как часть URL (вместо строки «:id»). Остальная информация передается в HTTP теле запроса как и в endpoint 3.1
   4.4 **DELETE /rest/user/:id** — endpoint для удаления юзера по ID. ID передается как часть URL (вместо строки «:id»)

   *** Усложненное задание:**
   4.a) Написать unit тесты для функций, предназначенных для общения с базой данных без поднятия базы данных

5. Путь **/rest/hash**

   Необходимо реализовать подсчет следующей хэш функции:
   1) Взять CRC64 хэш от входной строки
   2) Взять текущий timestamp с точностью до наносекунд
   3) Сделать логическое «И» текущего timestamp и текущего хэша
   4) Повторить шаги 2-3 в течение минуты с интервалом в 5 секунд
   5) Посчитать число единиц в двоичной записи полученного числа. Количество единиц и будет являться «хэшом»

   Более того нельзя заставлять клиента REST API сервера ждать. Поэтому у клиента есть возможность дать заявку на расчет данного хэша.

   Доступные endpoint'ы:
   5.1 **POST /rest/hash/calc —** endpoint для подачи заявки на расчет хэша. Данные для хэширования передаются в HTTP теле запроса. В ответ приходит уникальный идентификатор заявки.
   5.2 **GET /rest/hash/result/$id —** endpoint для получения результата для заявки с идентификатором $id. Команда возвращает «PENDING», если результат еще не посчитан и сам хэш, если подсчет уже произошел

*** Усложненное задание:**
5.a) Написать свою реализацию функции, извлекающей текущий timestamp, с учетом того, что в один момент времени ее может вызывать только один исполнитель.
5.b) Ограничить количество одновременно вычисляющихся хешей некой константой N

6. Путь **/rest/self (Дополнительное задание по желанию)**

   Необходимо реализовать поиск всех идентификаторов (имена переменных, констант, типов, функций и т. п.) в исходных файлах тестового задания, которые содержат заданную подстроку.
   Доступные endpoint’ы:
   6.1 **GET /rest/self/find/$str —** endpoint для поиска идентификаторов. Возвращает список всех идентификаторов, удовлетворяющих заданному условию.

   Для всех команд необходимо написать юнит тесты в отдельном файле.