Тестовое задание на вакансию Backend разработчик (Python)

Для каждого из двух представленных кейсов-примеров кода нужно провести ревью (review) кода. Ревью должно покрывать вопросы корректности, применимости, эффективности и читаемости кода. Приветствуются все замечания, даже самые мелкие. В результате мы ожидаем получить от вас перечисление мест кода с развернутым описанием проблем и путей их решения. В случае наличия нескольких путей решения, приветствуется их сравнение и объяснение в каких случаях какие варианты лучше.

Можете считать, что в кейсах используется Python 3.8.5. Ниже для кейсов также перечислены некоторые вводные данные.

Результат ревью можете оформить в удобном вам формате.

Также ниже перечислены дополнительные вопросы к кейсам, по которым нам интересно узнать ваши мысли. Включите их в результаты ревью (отдельно оформлять не нужно).

Case 1 - script

Вводные

1. Чтение фото, его поворот и изменение размера в process_image занимают в среднем 10 мс;
2. Функция calc_magic_number занимает в среднем 10мс;
3. Запрос внутри process_image занимает в среднем 30мс;
4. Функция send_stats занимает в среднем 10мс;

Дополнительные вопросы

1. Представьте, что ваш компьютер имеет бесконечное число ядер. Ограничено ли масштабирование скорости обработки изображений при росте числа используемых потоков? Если "да", то почему? Оцените максимальное число фото в секунду, которое сможет обрабатывать данный скрипт?
2. Было бы уместней для данного скрипта использовать вместо threading asyncio или multiprocessing? Или даже изменить принцип обработки? Если да, то почему и каким образом нужно переделать обработку?
3. Является ли для текущей реализации скрипта проблемой входной каталог, в котором будет 1 млн изображений? А 100 млн?

Case 2 - flask

Вводные

Сервис позволяет вести список заданий (Task) с привязкой к пользователям (User). Пользователь имеет доступ только к тем заданиям, которые он создал. Для пользователей хранится их адрес электронной почты. Адрес электронной почты должен быть уникален для пользователя.

Доступна функция, которая отдает пользователю не занятое на текущий момент задание и помечает его занятым. Одно задание не должно быть отдано в ответ на разные запросы.

При создании задания к нему прикрепляется картинка, которую потом можно получить отдельным запросом. Картинки хранятся на диске. При удалении задания картинка должна удаляться. Картинки могут иметь любой формат: jpeg, png, bmp и т.д. Также для задания сохраняется метаинформация, представляющая собой строку.

Для пользователя ведется счетчик созданных заданий, который увеличивается в момент создания задания. Удаление заданий не уменьшает счетчик, то есть он только растет. Также должна быть реализована доступная всем пользователям функция, которая возвращает сумму счетчиков всех пользователей.

Удаление пользователя должно приводить к удалению всех его заданий.

Краткое описание реализованного API:

- POST /users - создание пользователя; email передается в поле email в json; возвращается json с информацией о созданном пользователе;
- DELETE /user/:u_id - удаление пользователя с идентификатором u_id; удалять пользователь может только сам себя;
- GET /tasks - получение списка всех заданий пользователя;
- POST /tasks - создание задания; картинка передается в виде base64 строки в поле image в json, а метаинформация в поле meta в json;
- GET /tasks/:t_id - получение информации о задании с идентификатором t_id;
- GET /tasks/:t_id/image - получение картинки задания с идентификатором t_id;
- DELETE /tasks/:t_id - удаление задания с идентификатором t_id;
- GET /take_free_task - получение свободного задания;
- GET /total_tasks_created - получение суммы счетчиков созданных заданий всех пользователей;

Аутентификация осуществляется передачей query parameter user_id с идентификатором пользователя и является обязательной для всех методов, кроме создания пользователя.

Дополнительные вопросы

1. Есть ли у вас предложения по изменению API (например, по формату входных или выходных данных)?