Тестовое задание (1–2 часа): Мини-CRM распределения лидов между операторами по источникам

Цель:
Реализовать небольшой сервис на Python, который:

1. Хранит операторов и их загрузку.
2. Хранит лидов, которые могут обращаться из разных источников (ботов).
3. Хранит настройки распределения трафика по операторам для каждого источника.
4. Автоматически распределяет новые обращения лидов между операторами:
    - с учётом лимита нагрузки,
    - в процентном/весовом соотношении по заранее заданным числовым «компетенциям» (весам) по каждому источнику.

Структуру моделей и названия сущностей вы придумываете сами.

---

1. Технологический стек

Обязательно:

- Язык: **Python**
- Web-фреймворк: **FastAPI**
- ORM: **SQLAlchemy** (sync или async — на ваш выбор)
- База данных: **SQLite** (в файле или in-memory)

Docker, миграции, тесты — не обязательны, но будут плюсом, если вы успеете.

---

2. Предметная область (логика, без жёсткой схемы БД)

Нужно реализовать минимум следующие концепции (как именно оформить в моделях — на ваше усмотрение):

1. **Оператор**
    - Активен / не активен (может или не может получать новые обращения).
    - Имеет лимит по максимальному количеству «активных» лидов/обращений (формулировку и реализацию активности вы выбираете сами).

- Для каждого источника (бота), с которым он работает, имеет числовой **вес** (компетенцию / долю трафика).
  Пример: для источника A оператор1 — вес 10, оператор2 — вес 30 → примерно 25% и 75% трафика соответственно.

2. **Лид**
   - Представляет одного конечного клиента.
   - Один и тот же лид может написать через несколько разных ботов/источников.
   - Должен быть способ однозначно понять, что обращения относятся к одному и тому же лиду (например, общий внешний идентификатор, телефон, email и т.п. — на ваше усмотрение).
3. **Источник / бот**
   - Канал, из которого пришло обращение (один из условных «30 ботов»).
   - Для каждого источника задаётся конфигурация, какие операторы его обслуживают и с какими весами.
4. **Обращение / контакт**
   - Конкретный факт, что лид написал из определённого источника.
   - При создании обращения система должна:
     - связать его с лидом (найти существующего или создать нового),
     - определить источник,
     - выбрать и назначить оператора по правилам ниже,
     - сохранить результат.

---

3. Бизнес-логика распределения обращений

При создании **нового обращения от лида** система должна:

1. **Определить лида**
   - По входным данным найти существующего лида.
   - Если такой лид не найден — создать нового.
2. **Определить доступных операторов для источника**
   - Найти всех операторов, назначенных на данный источник (бот) в конфигурации.
   - Отфильтровать операторов по условиям:
     - оператор активен;
     - его текущая нагрузка не превышает лимит (по количеству активных лидов/обращений — на ваше усмотрение, главное описать в README, что именно вы считаете нагрузкой).
3. **Распределить с учётом весов (процентного соотношения)**

- У каждого оператора для этого источника есть числовой вес (например, 10, 20, 50 и т.д.).
- Нужно выбирать оператора таким образом, чтобы доля обращений в среднем соответствовала этим весам.
  Допустимые варианты:
  - случайный выбор с вероятностью `вес / сумма_весов` среди подходящих операторов;
  - детерминированный алгоритм, который по истории обращений старается выдерживать заданные доли.
- При этом оператор не должен превышать лимит нагрузки: если оператор «выиграл», но его лимит уже исчерпан, нужно выбрать другого подходящего или признать, что подходящих нет.

4. **Создать обращение**
   - Связать обращение с:
     - лидом,
     - источником,
     - назначенным оператором (если он найден).
   - Если подходящих операторов нет:
     - можно создать обращение без оператора,
     - либо вернуть 4xx-ошибку.
       Выберите один вариант и опишите его в README.

---

4. API (примерный набор операций)

Названия эндпоинтов и формат запросов/ответов вы придумываете сами, но должны быть реализованы операции следующего смысла:

1. **Управление операторами**
   - Создание оператора.
   - Просмотр списка операторов.
   - Управление лимитом нагрузки и активностью оператора.
2. **Настройка распределения по источникам**
   - Создание источника (бота).
   - Настройка для источника списка операторов и их весов (процентного распределения).
     Это может быть отдельный эндпоинт или часть CRUD по источнику/оператору.
3. **Регистрация обращения**

- Эндпоинт, принимающий данные:
  - идентификатор лида (или данные, по которым его можно определить/создать),
  - идентификатор источника (бота),
  - дополнительные данные обращения по вашему усмотрению.
- Внутри:
  - найти/создать лида,
  - выбрать оператора по описанным правилам,
  - создать обращение.
- В ответ вернуть информацию об обращении и назначенном операторе (если есть).

4. **Просмотр состояния**
   - Эндпоинты для просмотра:
     - списка лидов и их обращений,
     - распределения обращений по операторам и источникам (хотя бы в простом виде, чтобы было понятно, что один лид может иметь несколько обращений из разных источников).

---

5. Ожидаемый результат

Ожидаем от вас:

1. Архив или ссылку на репозиторий с:
   - кодом FastAPI-приложения,
   - моделями/слоями работы с БД,
   - реализацией логики распределения по весам.
2. Краткий `README.md`, где указано:
   - как запустить проект (минимум команда вида `uvicorn ...`),
   - краткое текстовое описание вашей модели данных (какие сущности и как связаны),
   - описание алгоритма распределения:
     - как определяется, что обращения принадлежат одному и тому же лиду,
     - как учитываются веса операторов по источникам,
     - как учитываются лимиты нагрузки,
     - что происходит, если подходящих операторов нет.

Тесты и Docker приветствуются, но не обязательны в рамках 1–2 часов.

---

6. Критерии оценки

В первую очередь будет оцениваться:

- Архитектура и нейминг сущностей (модели, связи, сервисы).
- Корректность реализации распределения с учётом весов и лимитов.
- Умение работать с БД и ORM.
- Структура и читаемость кода.
- Адекватность и предсказуемость API.

Старайтесь уложиться в **1–2 часа**. Лучше сделать немного, но ясно и аккуратно, чем пытаться покрыть максимальный объём за счёт качества.