Привет! Это Хабр Карьера и Хекслет

Перед вами второе тестовое. Оно основано на реальных задачах из вакансий для джунов в бэкенд-разработке.

Для выполнения тестового вам могут пригодиться эти курсы:

- **HTTP API**
- **Протокол HTTP**
- **Flask**

Совет: попробуйте сначала решить задание самостоятельно. Если где-то застопорились, заглядывайте в курсы. Если чувствуете, что задание сложновато, поставьте себе цель изучить этот курс в ближайшие дни, пока доступ к материалам Хекслет остаётся бесплатным.

А ещё, если вдруг пропустили первое тестовое или только что к нам присоединились, вот **ссылка на него**.

Описание задания

Реализуйте RESTful API, которое позволяет сохранять данные генетических тестов животных, а также выполнять подсчет агрегированной статистики. API должно поддерживать добавление данных, просмотр записей и вычисление статистики по видам животных.

Функциональность API

1. **Добавление данных генетического теста:**

- Пользователь может добавить запись о генетическом тесте с указанием животного, даты, продуктивности (milk_yield) и состояния здоровья.

2. **Просмотр записей**:

- Получение списка всех тестов.
- Фильтрация записей по виду животного.

3. **Подсчет**:

- Количества тестов для каждого вида животных.
- Среднего и максимального значения продуктивности.
- Процента животных с хорошим состоянием здоровья (health_status = "good").

Пример структуры базы данных

Таблица "Генетические тесты" (GeneticTests)

Поле	Тип	Описание
id	Integer	Уникальный идентификатор
animal_name	String	Имя животного
species	String	Вид (корова, свинья и т.д.)
test_date	Date	Дата проведения теста
milk_yield	Float	Продуктивность
health_status	String	Состояние здоровья ("good" или "poor")
created_at	Timestamp	Дата создания записи

Пример API

HTTP Метод	URL	Описание
POST	/tests	Добавить запись о генетическом тесте
GET	/tests	Получить список всех записей
GET	/tests?species=корова	Получить записи, отфильтрованные по виду
GET	/statistics	Получить агрегированную статистику

Пример работы API

1. **Добавление данных**:

```
POST /tests
Content-Type: application/json
{
"animal_name": "Буренка",
"species": "корова",
"test_date": "2023-11-18",
"milk_yield": 28.5,

"health_status": "good"
}
```

Ответ:

```
{
"message": "Данные успешно добавлены",
"id": 1
}
```

2. Получение всех записей

```
GET /tests
```

Ответ:

```
[
   {
      "id": 1,
      "animal_name": "Буренка",
      "species": "корова",
      "test_date": "2023-11-18",
      "milk_yield": 28.5,
      "health_status": "good"
   },
   {
      "id": 2,
      "animal_name": "Мурка",
      "species": "овца",
      "test_date": "2023-11-19",
      "milk_yield": 15,
      "health_status": "poor"
   }
]
```

3. Получение статистики:

```
GET /statistics
```

Ответ:

```
{
   "statistics": [
       {
           "species": "корова",
           "total_tests": 1,
           "avg_milk_yield": 28.5,
           "max_milk_yield": 28.5,
           "good_health_percentage": 100
       },
       {
           "species": "овца",
           "total_tests": 1,
           "avg_milk_yield": 15,
           "max_milk_yield": 15,
           "good_health_percentage": 0
       }
    ]
}
```

Требования

1. **Технологии**:

- Django REST Framework для реализации API.
- PostgreSQL для хранения данных.

2. **Функциональные требования**:

- Реализовать эндпоинты для описанных функций.
- Использовать миграции для создания базы данных.
- Обеспечить валидацию входных данных.

3. **Документация**:

- Добавьте README с инструкцией по запуску и примерами запросов.

4. **Обработка ошибок**:

- Корректно обрабатывать некорректные запросы (например, отсутствие обязательных полей).

Критерии оценки

1. **Работоспособность**: корректная обработка запросов и подсчет статистики.
2. **Качество кода**: структурированность, читаемость.
3. **Обработка ошибок**: API возвращает информативные сообщения об ошибках.
4. **Документация**: README с примерами запросов.