Тестовое задание на должность .NET бэкенд-разработчика

Используемые технологии

- .NET 8 (C# 12)
- REST API
- SQLite

Цель задания

В ходе данного тестового задания проверяются следующие навыки:

1. Умение работать с REST API.
2. Умение проектировать базы данных (БД далее).
3. Умение писать Unit-тесты.
4. Работа с асинхронными операциями *(`async`/`await`/`Task`)*.
5. Освоение новых технологий *(если придётся обучаться новому в ходе выполнения задания – указать отдельно: что было в новинку; при наличии трудностей – с чем они были связаны)*.

Исходный код созданных приложений следует выложить в систему контроля версий, допускающую открытый доступ к репозиторию, и дать на него ссылку. *Если по каким-то причинам это невозможно сделать – прислать **zip**-архив с исходным кодом, без скомпилированных и/или созданных в процессе работы приложений файлов.*

Задание

1. Создать веб-сервис на .NET 8, предоставляющий API с использованием Swagger *(выбор библиотеки на усмотрение соискателя)*. Данный сервис может работать со следующими двумя моделями:

   `Vehicle`: транспортное средство (ТС далее). Содержит поля:

   - Идентификатор ТС (`VehicleId`, тип `UUID`);
   - Название (`Name`, тип `string`, максимальная длина 30 символов).

   `Coordinate`: координаты, полученные с ТС. Содержит поля:

   - Широта (`Latitude`, тип `double`);
   - Долгота (`Longitude`, тип `double`);
   - Временная метка координат в секундах (`Timestamp`, тип `long`);
   - Идентификатор ТС (`VehicleId`, тип `UUID`).

   При запуске сервис очищает БД *(способ взаимодействия с БД – на усмотрение соискателя)* и генерирует в ней случайное количество ТС (числом от одного до ста; названия формируются случайным образом на усмотрение соискателя), и для каждого ТС создаёт случайные координаты (числом от одной до тысячи) с шагом временной метки в три секунды между координатами. Каждая координата в пределах одного ТС имеет уникальную временную метку (т.е. в БД не может быть две записи с одинаковым временем для одного и того же ТС).

   Данный сервис должен уметь обрабатывать три следующих запроса:

   - **`GET`** `/vehicles` – возвращает список всех ТС из БД *(`Vehicle[]`)*.

   - **`POST`** `/coordinates/find` – на вход подаётся массив идентификаторов ТС *(т.е. `Guid[]`)*, по которым метод должен найти в БД координаты и вернуть их *(т.е. `Coordinate[]`)*,

отсортированные сначала по **имени** ТС, а после по временной метке координаты (от большей к меньшей).

- **POST** `/coordinates/calculate_path` – на вход подаётся массив координат *(т.е. телом запроса является та же структура данных, что используется для ответа в методе* ***POST*** *`/coordinates/find`*). Данный метод должен рассчитать суммарное расстояние между принятыми на вход координатами для каждого ТС в метрах и милях, после чего вернуть рассчитанные значения, подписав результат каждого расчёта **именем** ТС *(для примера см. листинг 1 ниже)*.

```
{
  "Автобус А1":
  {
    "Metres": 2367.458,
    "Miles": 1.471
  },
  "Автобус А2":
  {
    "Metres": 108.305,
    "Miles": 0.067
  }
}
```

Листинг 1 – Пример успешного ответа от сервера на запрос POST /coordinates/calculate_path

2. Написать автоматизированные тесты для созданного в пункте 1 веб-сервиса *(с использованием NUnit или XUnit – на усмотрение соискателя)*. Как минимум надо учесть следующие ситуации под каждый метод:

   - **GET** `/vehicles`
     - ожидаемое поведение – получен в ответ не пустой массив ТС.

   - **POST** `/coordinates/find`
     - на вход подаётся массив с существующими в БД идентификаторами ТС (ожидаемое поведение – получен в ответ не пустой массив координат);
     - на вход подаётся массив с несуществующими в БД идентификаторами ТС (ожидаемое поведение – получен в ответ пустой массив координат).

   - **POST** `/coordinates/calculate_path`
     - на вход подаётся массив с длинной от одного и больше (ожидаемое поведение – получен ответ с рассчитанными значениями);
     - на вход подаётся пустой массив или `null` (ожидаемое поведение – получен пустой объект).

На что обратить внимание

1. Широта и долгота – это географические координаты, которые не могут превышать определённых значений.
2. Земля имеет форму эллипсоида, но для этого задания её радиус можно считать равным 6 371 000 метров.
3. Код должен быть читабельным: никаких участков кода наподобие `var objs = await get()`. В идеале – код должен читаться даже в блокноте без подсветки синтаксиса и подсказок *(явные указания типов, понятные названия переменных…)*.
4. Т.к. целевой платформой указан .NET 8, ожидается применение современных возможностей C# *(где это возможно и имеет смысл по мнению соискателя)*.