ubiwhere

Exercício Python & Django

Contexto

Este exercício de admissão está dividido em 3 partes. A **Parte 1** é obrigatória, a **Parte 2** é opcional mas recomendada, e a **Parte 3** é opcional. Recomendamos a leitura do documento até ao fim antes de começar a resolução do exercício.

Introdução

Pretende-se que seja desenvolvida uma *API REST,* utilizando *Django Rest Framework,* com o principal objetivo de alimentar uma *dashboard* de monitorização de tráfego rodoviário. Para tal, esta *API* deve ser capaz de fornecer a informação necessária para a tomada de decisões relativamente à intensidade de tráfego registada em segmentos de estrada locais.

A monitorização do tráfego é feita relativamente à representação geográfica de segmentos de estrada, e para cada um deles deve ser tida em conta a seguinte informação:

- Velocidade média dos veículos;
- Caracterização da intensidade de tráfego (inferida a partir da velocidade média);
- Data e hora do registo dos valores anteriores.

Velocidade média	Intensidade
≤ 20 km/h	elevada
> 20 e ≤ 50 km/h	média
> 50 km/h	baixa

Tabela 1 - Caracterização da intensidade do tráfego de acordo com a velocidade média registada

Nota: Assume-se que os *intervalos* de velocidade média, apresentados na Tabela 1, podem sofrer alterações no futuro, e como tal a caracterização atribuída a cada leitura deve respeitar sempre os *intervalos* mais recentes (mesmo para as leituras já existentes na base de dados).

Além de permitir as operações *CRUD* (*create*, *read*, *update* e *delete*), a *API* deve controlar que operações é que cada tipo de utilizador pode realizar sobre os segmentos de estrada, tendo em conta a seguinte tabela.

Tipo de Utilizador	Operações
Administrador	Create, Read, Update, Delete
Anónimo	Read

Tabela 2 - Permissões de operações autorizadas por tipo de utilizador

De forma a facilitar a população da base de dados, podes encontrar aqui uma fonte de dados que poderás importar da forma que pretendes (documentando o processo preferencialmente). Desta forma, quando partilhares a tua solução, a *API* estará imediatamente disponível para devolver informação. Caso tenhas algum problema a aceder ao ficheiro, por favor avisa para resolvermos a situação.

Parte 1

- Desenvolver uma *API REST* que:
  - Permita criar, ler, editar e eliminar segmentos de estrada tendo em conta o tipo de utilizador (de acordo com as permissões da Tabela 2);
  - Permita criar, ler, editar e eliminar leituras de velocidade média para um determinado segmento de estrada tendo em conta o tipo do utilizador (de acordo com as permissões da Tabela 2);
  - Para cada segmento de estrada, devolva o número total de leituras associadas ao segmento de estrada em questão.
- Integrar documentação interativa da *API* (*Swagger* ou solução semelhante). Preferencialmente, esta documentação deve ser disponibilizada no endereço "`/api/docs`".
- Carregar os dados fornecidos para a base de dados de forma a serem disponibilizados pela *API*.
- Deverá ser possível efetuar a gestão de utilizadores via Django Admin.

Notas:

- A componente da *API REST* deve ser desenvolvida com *Django* (para expor uma *API* a partir de um projeto *Django*) sendo recomendada a utilização da package *Django Rest Framework*.
- Deves fornecer instruções de como inicializar o projeto *Django* com a *API*. Caso tenhas conhecimentos em Docker, podes fazê-lo através de Docker containers (utilizando docker-compose). O mesmo se aplica para outros serviços, como a base de dados que utilizes para estruturar o teu projeto.

- A base de dados deixamos ao teu critério (*SQLite, PostgresQL, MySQL, etc*). Mas recomendamos *PostgreSQL* com *PostGIS* para recursos geoespaciais.
- Incentivamos o uso do formatador de código black, mas poderás usar outro à tua escolha.
- Não é obrigatório, mas valorizamos a utilização do Github como repositório do código fonte a partilhar connosco.

Parte 2

- Criar testes unitários para validar as várias funcionalidades da *API* e as permissões de utilizador.
- Adicionar uma funcionalidade que permita filtrar os segmentos de estrada tendo em conta a caracterização (elevada, média, baixa) da última leitura realizada.

Parte 3

Pretende-se agora proceder à integração da informação de um sistema de sensores de tráfego. Estes sensores são dotados de uma câmera que captura todas as passagens de veículos nos segmentos de estrada, registando a matricula e a data e hora de observação. Estes sensores são de natureza móvel, pelo que serão colocados de forma ocasional e rotativa nos vários segmentos de estrada.

Cada sensor irá acumular um determinado número de registos, e depois enviará estes registos em *bulk* para a plataforma via pedido *HTTP POST*. Abaixo encontras um exemplo de pedido que será enviado pelo sensor:

```
[
  {
    "road_segment" : 1,
    "car__license_plate":"AA16AA",
    "timestamp": "2023-05-29T09:27:26.769Z",
    "sensor__uuid": "270e4cc0-d454-4b42-8682-80e87c3d163c"
  },
  {
    "road_segment" : 2 ,
    "car__license_plate":"BB17BB",
    "timestamp": "2023-05-29T17:05:21.713Z",
    "sensor__uuid": "a3e86bd0-c19f-44e9-84c0-eadf4d4da197"
  }
]
```

O campo **road_segment** indica o ID do segmento de estrada. O campo **car__license_plate** indica a matrícula do carro observado. O **timestamp** indica a data de observação. O **sensor__uuid** indica o identificador único universal do sensor que realizou a observação.

Desenvolve um *endpoint* que permita aos administradores da plataforma especificar uma matrícula de um carro via parâmetro, e juntamente com as informações do carro, devolva todas as passagens registadas nas últimas 24 horas, incluindo a informação do sensor e a informação do segmento de estrada.

Notas:

- Assume-se que todos os sensores estão previamente registados na base de dados, e têm um identificador único universal que corresponde ao **sensor__uuid** enviado no pedido. Podes encontrar a lista de sensores a importar neste link.
- Por sua vez, os carros não se encontram previamente registados na base de dados, pelo que deve ser feito esse registo sempre que é recebida uma matrícula nova. Para além da matrícula, deve ser guardada a data e hora atual do sistema quando é efetuado um registo de um novo carro.
- Assume-se que os sensores comunicam com a plataforma via *HTTP*, e devem autenticar-se com ela através de um sistema de *API Keys*. Deves considerar que a *API key* necessária é **"23231c7a-80a7-4810-93b3-98a18ecfbc42"**. Pedidos que não incluam esta chave devem ser considerados inválidos e rejeitados. Fica ao teu critério a maneira como os sensores enviam esta chave no pedido.

Links Úteis

- Documentação *Django*: https://www.djangoproject.com/start/
- Django Rest Framework: https://www.django-rest-framework.org/
- Docker: https://docs.docker.com/get-started/ e https://docs.docker.com/compose/gettingstarted/
- Localização geográfica: django-rest-framework-gis/README.rst at master · openwisp/django-rest-framework-gis
- Imagem Docker de PostGIS: https://hub.docker.com/r/mdillon/postgis/
- Documentação da API: https://drf-spectacular.readthedocs.io/en/latest/
- *Package* de filtros: https://django-filter.readthedocs.io/en/stable/
- Debug e otimização: https://github.com/jazzband/django-debug-toolbar

Critérios de avaliação

1. Funcionalidade: o exercício responde aos requisitos funcionais especificados e todas as funcionalidades solicitadas estão implementadas corretamente.

2. Boas práticas de desenvolvimento: a estrutura do código fonte do projeto está bem organizada, com uma divisão adequada em módulos, classes e funções. A utilização de nomenclatura adequada de variáveis e funções, e de comentários explicativos, assim como a legibilidade e a clareza do código são consideradas.

3. Qualidade do código: avaliada em termos de eficiência, escalabilidade, segurança e robustez. É verificado se foram aplicadas técnicas adequadas para lidar com exceções, validação de entrada de dados, tratamento de erros e segurança da aplicação.

4. Uso adequado do Django: és avaliado/a por teres utilizado corretamente os recursos, funcionalidades e melhores práticas do Django e Django Rest Framework.

5. Eficiência e desempenho: o código é eficiente e otimizado relativamente a queries à base de dados.

6. Arquitectura da solução: design da API segundo conceitos REST e modelação da base de dados.

7. Documentação: documentação da API, documentação do repositório, e instruções de configuração e utilização do projeto.

Todas as dúvidas que surjam podes partilhar connosco, afinal de contas, caso venhas para a Ubiwhere, estarás a trabalhar em equipa e não sozinho.

Bom trabalho!