RELATÓRIO DE API GERENBUS – MATHEUS PEREIRA BARROS

TÓPICOS ESPECIAIS DE PROGRAMAÇÃO I – ADS V

1. A API deve possuir pelo menos 4 entidades relevantes e relacionadas via mapeamento objeto relacional.

Foram criadas seis entidades relativas a um sistema de gerenciamento de uma empresa de veículos de transporte terrestre, mais especificamente ônibus (o que não exclui outros tipos de veículos), em que são as seguintes entidades: Garagem(models.Model), Onibus(models.Model), Rota(models.Model), Funcao(models.Model), Funcionario(models.Model) e Viagem(models.Model).

Um Ônibus possui uma marca, modelo, placa e pertence a uma Garagem, recebendo o id da mesma como chave estrangeira, Uma Garagem possui endereço e quantidade de vagas. Uma Rota conterá o trecho de uma Viagem e terá um nome, um destino e um ponto de partida, um horário, um dia da semana e o valor da passagem. Uma Viagem possui um horário, quantidade de passageiros, uma rota, um ônibus e um funcionário, recebendo as chaves de Rota, Onibus e Funcionario. Um Funcionário possui nome, CPF e uma função. Uma Função possui uma descrição relativa a mesma.

2. Pelo menos uma entidade deve ser integrada ao esquema de autenticação do Django.

A entidade Funcionario está integrada ao esquema de autenticação do Django, possuindo um atributo user de django.contrib.auth.models no arquivo empresa/models.py

empresa/models.py:

```
from django.contrib.auth.models import User
class Funcionario(models.Model):
  name = models.CharField(max_length=128)
  user = models.OneToOneField(User, related_name='user')
```

3. Parte da API deve ser somente leitura e parte deve ser acessível apenas para usuários autenticados.

A API possui rotas que são apenas somente leitura. Outras rotas são vistas apenas por usuários autenticados. A view de usuários UserList(generics.ListAPIView) e de rotas rotas_list(generics.ListAPIView) são apenas somente leitura. As views de listagens, como funcoes_list(generics.ListCreateAPIView) e rotas_list(generics.ListCreateAPIView) podem ser alteradas usuários autenticados, utilizando método POST. As views de detalhes, que em suas rotas recebem como parâmetro o ID da entidade, como ônibus_detail(generics.RetrieveUpdateDestroyAPIView) e viagens_detail(generics.RetrieveUpdateDestroyAPIView) permitem a usuários autenticados o uso de métodos como PUT e DELETE.

As views podem ser definidas como somente leitura através das views rest_framework.generics.ListAPIView, como leitura e uso do método PUT através de rest_framework.generics.ListCreateAPIView e controle total através da view rest_framework.generics.RetrieveUpdateDestroyAPIView. Apenas usuários autenticados podem fazer alterações de tal relevância nos dados graças a permissões contidas no arquivo permissions.py além das permissões que o rest_framework dá como opção ao desenvolvedor através do atributo permission_classes em cada view.

empresa/views.py:

```
class onibus_list(generics.ListCreateAPIView):
  queryset = Onibus.objects.all()
  serializer_class = OnibusSerializer
  name = 'onibus-list'
  permission_classes = (permissions.IsAuthenticatedOrReadOnly,)
```

4. A API deve ser documentada com Swagger ou alguma outra sugestão da página: http://www.django-rest-framework.org/topics/documenting-your-api/.

A presente API foi documentada com Swagger, disponível na API Root e na rota localhost:8000/swagger. Após instalada no ambiente Django onde a API está localizada, foi necessário realizar configurações nos arquivos gerenbus/settings.py e empresa/urls.py como descrito pelo desenvolvedor do Swagger.

empresa/urls.py:

```
from rest_framework_swagger.views import get_swagger_view
schema_view = get_swagger_view(title='GerenBus')
```

```
urlpatterns = [
        url(r'^swagger/$', schema_view, name='swagger'),
]
```

gerenbus/settings.py:

```
INSTALLED_APPS = [
    'rest_framework_swagger',
]
```

5. Definir e usar critérios de paginação e Throttling. Esse último deve diferenciar usuários autenticados de não autenticados.

Foi definido o limite de cinco itens por página no arquivo gerenbus/settings.py. Além disso, a taxa de requisições para usuários anônimos foi limitada em 30 por hora e para usuários autenticados em 150 requisições na variável REST_FRAMEWORK:

gerenbus/settings.py:

```
REST_FRAMEWORK = {
'DEFAULT_PAGINATION_CLASS':
        'rest_framework.pagination.LimitOffsetPagination',
        'PAGE_SIZE': 5,
'DEFAULT_THROTTLE_CLASSES': (
        'rest_framework.throttling.AnonRateThrottle',
        'rest_framework.throttling.UserRateThrottle',
        ),
'DEFAULT_THROTTLE_RATES': {
        'anon': '30/hour',
        'user': '150/hour',
        },
    }
```

6. Implementar para pelo menos 2 entidades: filtros, busca e ordenação.

Foram aplicados filtros, busca e ordenação nas entidades Rotas e Funcionários. Após a instalação do Filter Backend via PIP utilizando pip install django-filter, é necessário configurar a presença do mesmo no arquivo gerenbus/settings.py.

gerenbus/settings.py:

```
REST_FRAMEWORK = {
```

```
    'DEFAULT_FILTER_BACKENDS':
        ('django_filters.rest_framework.DjangoFilterBackend',)
}
```

É necessário ainda importar o método filters no arquivo empresa/views.py:

empresa/views.py:

```
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework import filters, status
```

Nas views, é necessário declarar os atributos que serão filtrados, buscados e ordenados utilizando o atributo filter_backends()

empresa/views.py:

```
class funcionarios_list(generics.ListCreateAPIView):
  filter_backends = (filters.SearchFilter, filters.OrderingFilter, DjangoFilterBackend)
  ordering_fields = ('nome',)
  search_fields = ('nome',)
  filter_fields = ('nome',)

class rotas_list(generics.ListAPIView):
  filter_backends = (filters.SearchFilter, filters.OrderingFilter, DjangoFilterBackend)
  ordering_fields = ('nomeRota',)
  search_fields = ('nomeRota',)
  filter_fields = ('nomeRota',)
```

7. Criar testes unitários e de cobertura.

Foram criados testes unitários e de cobertura para verificar se as requisições realizadas nos testes retornam os códigos de status esperados.

empresa/tests.py:

```
def test_client_list(self):
    response = self.client.get(reverse('viagens-list'))
    self.assertEqual(response.status_code, status.HTTP_200_OK)
```

Os testes são executados com o comando python manage.py test na pasta ProjectWebAPI.