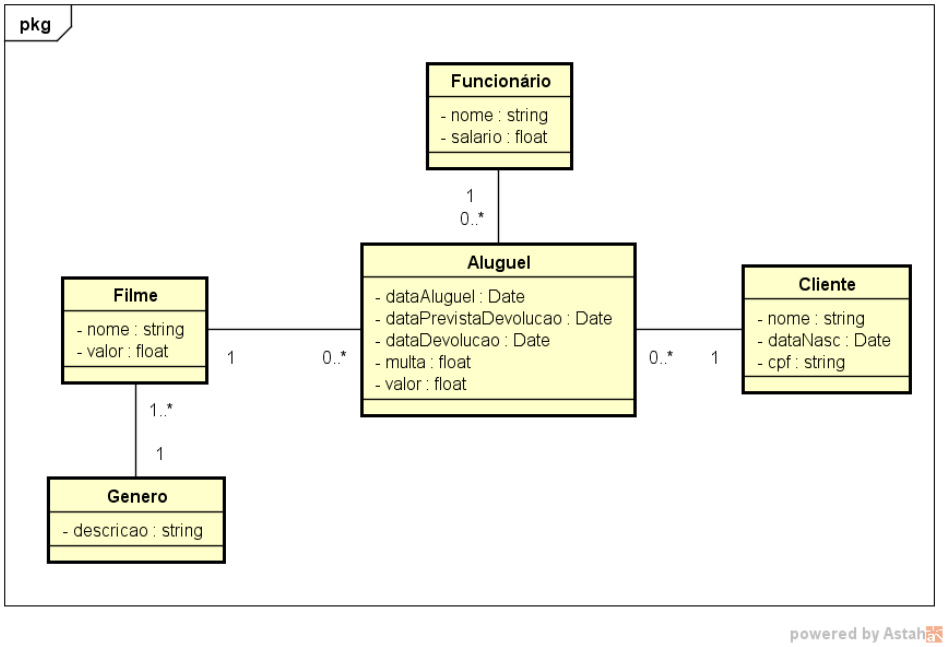
1. **A API deve possuir pelo menos 4 entidades relevantes e relacionadas via mapeamento objeto relacional:**
    - O sistema de aluguel de filmes possui 5 classes que são: gênero, filme, aluguel, funcionário e cliente, estas classes estão relacionadas conforme o diagrama a seguir:



2. **Pelo menos uma entidade deve ser integrada ao esquema de autenticação do Django:**
    - A entidade Funcionario é integrado com o sistema de autenticação do Django através do relacionamento que ela possui com o User do Django.

aluguel/models.py

```
1.  class Funcionario(models.Model):
2.      nome = models.CharField(max_length=255)
3.      user = models.oneToOneField(User, related_name='funcionario', on_delete=mo
    dels.CASCADE)
4.      salario = models.FloatField()
5.  ...
```

3. **Parte da API deve ser somente leitura e parte deve ser acessível apenas para usuários autenticados:**
    - As entidades Genero, Filme, User, Funcionarios são somente leitura e apenas os usuários autenticados podem fazer alterações, já as entidade Cliente e Aluguel são acessíveis apenas para os usuários autenticados.
    - Para isso foi adicionado na views.py da aplicação aluguel as permissões através do permissions do *rest_framework.permissions* e permissões personalizadas do *permissions.py*.

aluguel/views.py

```
class UserList(generics.ListCreateAPIView):
    ...
    permission_classes = (
        permissions.IsAuthenticatedOrReadOnly,IsUserOrReadOnly)

...

class AluguelDetail(generics.RetrieveUpdateDestroyAPIView):
    ...
    permission_classes = (permissions.IsAuthenticated,)
```

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 API foi documenta utilizado o Swagger, para isso foi preciso instalar o Swagger utilizando o comando *pip install django-rest-swagger* no terminal do computador e depois foi adicionado ao INSTALLED_APPS do *settings.py* e a rota da aplicação no arquivo *urls.py* do projeto.

apiAluguel/settings.py

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

apiAluguel/urls.py

```
schema_view = get_swagger_view(title='Aluguel API')

urlpatterns = [
    ...
    url(r'^swagger$', schema_view)
]

```

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

   - Toda a aplicação está configurada para exibir 5 itens por página, já a requisição ficou da seguinte forma, usuários não autenticado podem fazer 50 requisições, usuários autenticados podem fazer 100 requisições.
   - A paginação e o Throtting foram configuradas no *settigs.py* do projeto.

apiAluguel/settings.py

```
REST_FRAMEWORK = {
    ...
    'DEFAULT_THROTTLE_CLASSES': (
        'rest_framework.throttling.AnonRateThrottle',
        'rest_framework.throttling.UserRateThrottle',

    ),
```

```
    'DEFAULT_THROTTLE_RATES': {
        'anon': '50/hour',
        'user': '100/hour',
    },
    ...
}
```

6. **Implementar para pelo menos 2 entidades: filtros, busca e ordenação:**
   Filtro, busca e ordenação foram feitos paras as entidades, User, Funcionairo e Cliente para isso precisou instalar o Django Filter através do comando *pip install django-filter* feito no terminal do computador, e depois precisou ser referenciado no arquivo de configuração do projeto e inserido na *view.py* da aplicação.

apiAluguel/settings.py

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

aluguel/views.py

```
class UserList(generics.ListCreateAPIView):
    ...
    filter_backends = (filters.SearchFilter, filters.OrderingFilter,)
    search_fields = ('username', 'email')
    ordering_fields = ('username')
    ...
...
class FuncionarioList(generics.ListCreateAPIView):
    ...
    filter_backends = (filters.SearchFilter, filters.OrderingFilter,)
    search_fields = ('nome',)
    ordering_fields = ('nome',)
    ...
...
class ClienteList(generics.ListCreateAPIView):
    ...
    filter_backends = (filters.SearchFilter, filters.OrderingFilter,)
    search_fields = ('nome',)
    ordering_fields = ('nome', 'dataNasc')
    ...
...
```

7. **Criar testes unitários e de cobertura:**

Os testes foram adicionados ao arquivo tests.py da aplicação aluguel, para isso foi preciso salvar um arquivo json dos dados que estão no banco de dados através do comando *python manage.py dumpdata -o fixtures.json --exclude=contenttypes* no terminal. E para rodar os testes basta usar o comando *python manage.py test aluguel.tests.*

aluguel/tests.py

```
class APITest(TestCase):
  ...
  def test_user_list(self):
    response = self.client.get(reverse('user-list'))
    self.assertEqual(response.status_code, status.HTTP_200_OK)
```