

Relatório da API e Cliente

Nome: Elke Macline de Oliveira Sabino

Disciplina: Tópicos Especiais de Programação

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

R: As entidades usadas no projeto foram: Usuario, Autor, Livro, Estante e Operacao. O sistema baseia-se em troca e compra de livros entre usuários de diversas regiões.

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

R: Na api, a entidade Usuário está integrada com o User do Django, ao cadastrar um usuário, durante o cadastro são realizadas duas ações é criado um User no Django e após isso, é criado o usuário inserindo uma chave estrangeira OneToOneField. No arquivo Models pode ser verificado e a implementação do cadastro está na Views

Views linha 57:

```

57 class UsuarioList(generics.ListCreateAPIView):
58     queryset = Usuario.objects.all()
59     serializer_class = UsuarioSerializer
60     name='usuario-list'
61
62     def perform_create(self, serializer):
63
64         dados = self.request.data
65
66         if self.request.POST:
67             user = self.request.POST.get('user.username')
68             email = self.request.POST.get('user.email')
69             password = self.request.POST.get('user.password')
70         else:
71
72             user = dados['user']['username']
73             email = dados['user']['email']
74             password = dados['user']['password']
75
76         userserializer = UserSerializer(data={'username':user,
77                                             'email':email,
78                                             'password':password})
79
80
81         if userserializer.is_valid():
82             usar = userserializer.save()
83             serializer.save(user=usuar)
84

```

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

R: Esse requisito também foi realizado, algumas ações como cadastramento de livros, cadastramento de autores, efetuação de transações entre usuários, verificação das transações realizadas, só ocorrem se o usuário estiver autenticado

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

R: A documentação foi feita usando o Swagger, abaixo alguns prints do que fora feito:

No settings em APPs:

```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'livros',
    'crispy_forms',
    'rest_framework_swagger',
    'rest_framework.authtoken',
]

```

Resultado na API:

The image shows the Swagger API documentation interface. At the top, there is a green header with the Swagger logo, the text "Hello, ely", and buttons for "Django Logout" and "Authorize". Below the header, the word "documentacao" is displayed. A table lists the API endpoints, each with a "Show/Hide" button, a "List Operations" button, and an "Expand Operations" button. The endpoints listed are: api-token-auth, autor, estante, livro, operacao, transacao, user, and usuario. At the bottom, there is a field for "BASE URL:".

Endpoint	Show/Hide	List Operations	Expand Operations
api-token-auth	Show/Hide	List Operations	Expand Operations
autor	Show/Hide	List Operations	Expand Operations
estante	Show/Hide	List Operations	Expand Operations
livro	Show/Hide	List Operations	Expand Operations
operacao	Show/Hide	List Operations	Expand Operations
transacao	Show/Hide	List Operations	Expand Operations
user	Show/Hide	List Operations	Expand Operations
usuario	Show/Hide	List Operations	Expand Operations

[BASE URL:]

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

R: Este requisito também foi realizado, em settings na linha 63 podemos ver as configurações realizadas.

```

47 REST_FRAMEWORK = {
48
49     'DEFAULT_PAGINATION_CLASS':
50     'rest_framework.pagination.LimitOffsetPagination',
51     'PAGE_SIZE': 5,
52     'DEFAULT_FILTER_BACKENDS': (
53         'rest_framework.filters.DjangoFilterBackend',
54         'rest_framework.filters.SearchFilter',
55         'rest_framework.filters.OrderingFilter',
56     ),
57
58     'DEFAULT_AUTHENTICATION_CLASSES': (
59         'rest_framework.authentication.BasicAuthentication',
60         'rest_framework.authentication.SessionAuthentication',
61     ),
62
63     'DEFAULT_THROTTLE_CLASSES': (
64         'rest_framework.throttling.AnonRateThrottle',
65         'rest_framework.throttling.UserRateThrottle',
66     ),
67     'DEFAULT_THROTTLE_RATES': {
68         'anon': '200/hour',
69         'user': '200/hour',
70         'estantes': '200/hour',
71     }

```

Na views, também foi implementado de forma personalizada a paginação para classe de autores.

```

23 class LargeResultsSetPagination(PageNumberPagination):
24     page_size = 100
25     page_size_query_param = 'page_size'
26     max_page_size = 1000
27

```

```

183 class AutorList(generics.ListCreateAPIView):
184
185     queryset = Autor.objects.all()
186     pagination_class = LargeResultsSetPagination
187     serializer_class = AutorSerializer
188     name = 'autor-list'
189     DjangoFilterBackend.template='rest_framework/filters/django_filter.html'
190
191
192     permission_classes = (
193         permissions.IsAuthenticatedOrReadOnly,
194         IsOwnerOrReadOnly,
195     )
196
197
198     filter_fields = ('autor',)
199     search_fields = ('^autor',)
200     ordering_fields = ('autor',)
201
202     def perform_create(self, serializer):
203
204         usuario = Usuario.objects.get(user=self.request.user)
205         serializer.save(usuario=usuario)
206
207

```

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

R: Esse requisito foi realizado nas classes de Livros e Autores, onde podem ser feitas buscas, ordenação e filtrar.

Foi utilizado a importação `from rest_framework.filters import DjangoFilterBackend`

Para a utilização dos filtros, na views pode ser constatado na linha 109, os campos escolhidos para filtragem, como mostra a figura abaixo:

```

96 class LivroList(generics.ListAPIView):
97     queryset = Livro.objects.all()
98     serializer_class = LivroSerializer
99     name = 'livro-list'
100
101     DjangoFilterBackend.template='rest_framework/filters/django_filter.html'
102
103     permission_classes = (
104         permissions.IsAuthenticatedOrReadOnly,
105         IsOwnerOrReadOnly,
106     )
107
108
109     filter_fields = ('usuario',
110                     'autor',
111                     'titulo',
112                     'categoria',
113                     'ano_publicacao',)
114     search_fields = ('^titulo',)
115     ordering_fields = ('titulo',
116                      'ano_publicacao',)
117
118

```

Na classe AutorList também é outro exemplo onde o filtro foi usado, como mostra abaixo

```

183 class AutorList(generics.ListCreateAPIView):
184
185     queryset = Autor.objects.all()
186     pagination_class = LargeResultsSetPagination
187     serializer_class = AutorSerializer
188     name = 'autor-list'
189     DjangoFilterBackend.template='rest_framework/filters/django_filter.html'
190
191
192     permission_classes = (
193         permissions.IsAuthenticatedOrReadOnly,
194         IsOwnerOrReadOnly,
195     )
196
197
198     filter_fields = ('autor',)
199     search_fields = ('^autor',)
200     ordering_fields = ('autor',)
201
202     def perform_create(self, serializer):
203
204         usuario = Usuario.objects.get(user=self.request.user)
205         serializer.save(usuario=usuario)
206

```

OBS: No lado Cliente, feito em java, quando é aberto o formulário de cadastro de livros, o combobox com os nomes dos autores foi também usado o recurso do filtro através de uma requisição para que o mesmo recebesse os autores em ordem alfabética

7 - Criar testes unitários e de cobertura;

R: Os testes que foram feitos foram apenas usando um arquivo externo com a biblioteca Requests, para fazer as requisições e verificar se tudo estava ocorrendo bem, como mostra

abaixo uma parte do arquivo em imagem, sendo que o arquivo original com as requisições está no repositório com a API e o cliente

```
return status_code, text(resultado.status_code) if resultado.text

def cadastrarAutor(url, autor):
    corpo = json.dumps({'autor': autor})
    headers = {'content-type': 'application/json'}
    resultado = requests.post(url, data=corpo, headers=headers, auth=('francisco', 'francisco123'))

    return 'status_code: ' + str(resultado.status_code) + '\n' + resultado.text

def cadastrarLivro(url, dados):
    corpo = json.dumps(dados)
    headers = {'content-type': 'application/json'}
    resultado = requests.post(url, data=corpo, headers=headers, auth=('francisco', 'francisco123'))

    return 'status_code: ' + str(resultado.status_code) + '\n' + resultado.text

def cadastrarOperacao(url, dados):
    corpo = json.dumps(dados)
```

Observações finais do relatório:

O cliente foi feito usando linguagem java com algumas bibliotecas importantes como Jersey e GSON, foram criados os formulários usando o swing.

No cliente é possível fazer as seguintes operações:

- Efetuar o cadastro de um novo usuário
- Efetuar um login, permanecendo conectado, como uma sessão, foi implementado uma classe sessão em que o cliente se comunica com a API verificando o usuário e senha, confirmando o login no lado cliente é exibido o nome do usuário na tela e dessa forma o botão do LOGOF fica ativado para quando o usuário quiser encerrar a sessão.
- Pode ser realizado uma consulta verificando todos os livros cadastrados no sistema
- Pode ser realizado uma consulta verificando autores
- Pode ser realizado cadastros de novos livros, sendo exigido a autenticação, caso contrário retorna uma mensagem.
- É possível verificar os livros do usuário que está logado, que o mesmo cadastrou.
- É possível também excluir livros do usuário logado no sistema
- É possível fazer a consulta de todas as estantes no sistema verificando a disponibilidade das mesmas para compra ou troca
- É possível efetuar uma transação escolhendo o livro que o usuário logado se interessa, assim a estante deixa de ser listada como disponível
- O usuário pode a qualquer tempo se estiver logado verificar as suas transações realizadas.