

Django

TomerBu

נושאים להיום:

חזרה על מסקנות מהשיעור הקודם
סיכום מודלים בבלוג
עבודה עם מסד נתונים חיצוני

לפוסט יש תגיות:

תאמה של ממשק המשתמש והסריליזציה לעבודה עם Taggit
הציג תగיות באופן רצוי
Serializers
models
סריליזציה של קשרי גומלין
ולידציה

עדכו הViews בפרויקט:

המחלקה ModelViewSet יוצרת עבורנו את כל פעולות CRUD הסטנדרטיות

```
from .serializers import *
from .models import Comment, Post, UserProfile, PostUserLikes
from rest_framework.viewsets import ModelViewSet

class CommentsViewSet(ModelViewSet):
    queryset = Comment.objects.all()
    serializer_class = CommentSerializer

class PostsViewSet(ModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer

class UserProfileViewSet(ModelViewSet):
    queryset = UserProfile.objects.all()
    serializer_class = UserProfileSerializer

class LikesViewSet(ModelViewSet):
    queryset = PostUserLikes.objects.all()
    serializer_class = PostUserLikesSerializer
```

אבל... לkoחות רוצים לעצב את הAPI באופן שונה מברירת המחדל
לפחות מספר מתחזות שהם רוצים בהן שינוי
ולכן למדנו את המבנה השלם של המחלקות הפנימיות

עדכו הUrls בפרויקט:

המחלקה ModelViewSet יוצרת עבורנו את כל פעולות CRUD הסטנדרטיות

```
from .serializers import *
from .models import Comment, Post, UserProfile, PostUserLikes
from rest_framework.viewsets import ModelViewSet

class CommentsViewSet(ModelViewSet):
    queryset = Comment.objects.all()
    serializer_class = CommentSerializer

class PostsViewSet(ModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer

class UserProfileViewSet(ModelViewSet):
    queryset = UserProfile.objects.all()
    serializer_class = UserProfileSerializer

class LikesViewSet(ModelViewSet):
    queryset = PostUserLikes.objects.all()
    serializer_class = PostUserLikesSerializer
```

עדכון הUrls בפרויקט:

blog/urls.py

עבודה עם ראותר: יש לרשום כל ViewSet פעם אחת בלבד - הראותר יסיק את הכתובות הנכונות

```
from django.urls import path, include
from .views import CommentsViewSet, PostsViewSet, UserProfileViewSet, LikesViewSet

from rest_framework.routers import DefaultRouter

router = DefaultRouter()

router.register('comments', CommentsViewSet, basename='comment')
router.register('posts', PostsViewSet, basename='posts')
router.register('user-profile', UserProfileViewSet, basename='user-profile')
router.register('likes', LikesViewSet, basename='likes')

urlpatterns = [
    path('', include(router.urls)),
]
```

נראה יחד שນצטרכן להשתמש גם בViews Function Based Views
ועוד לפחות הפניות לדרישות הפרויקט

lec4/urls.py

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include('blog.urls')),
]
```

python manage.py runserver

http://127.0.0.1:8000/api/v1/

הVIEW Root API

מגיש מובנה עם הראوتر

Django REST framework

Api Root

Api Root

OPTIONS

GET



The default basic root view for DefaultRouter

GET /api/v1/

HTTP 200 OK

Allow: GET, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{  
    "comments": "http://127.0.0.1:8000/api/v1/comments/",  
    "posts": "http://127.0.0.1:8000/api/v1/posts/",  
    "user-profile": "http://127.0.0.1:8000/api/v1/user-profile/",  
    "likes": "http://127.0.0.1:8000/api/v1/likes/"  
}
```

Browsable API

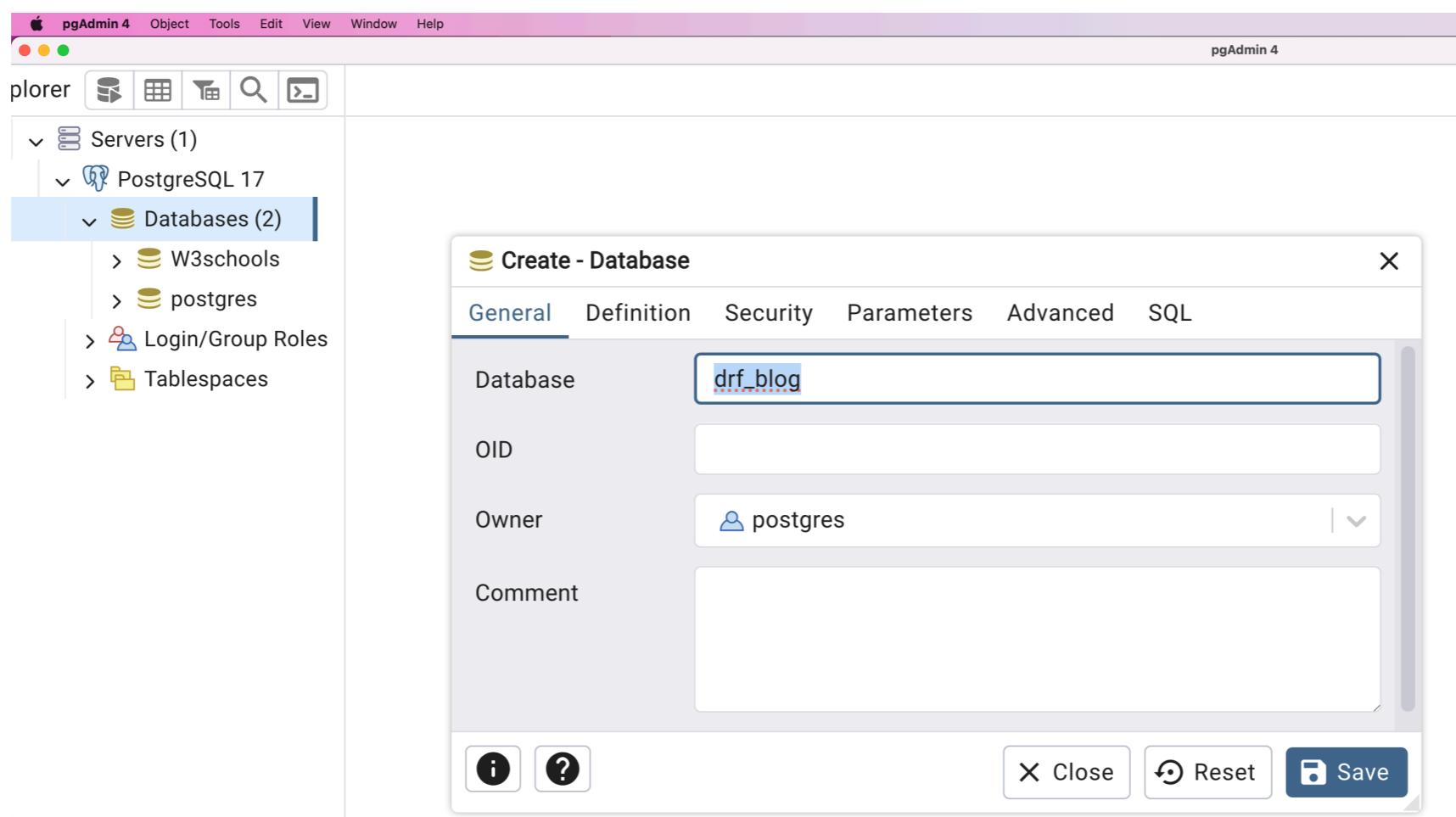
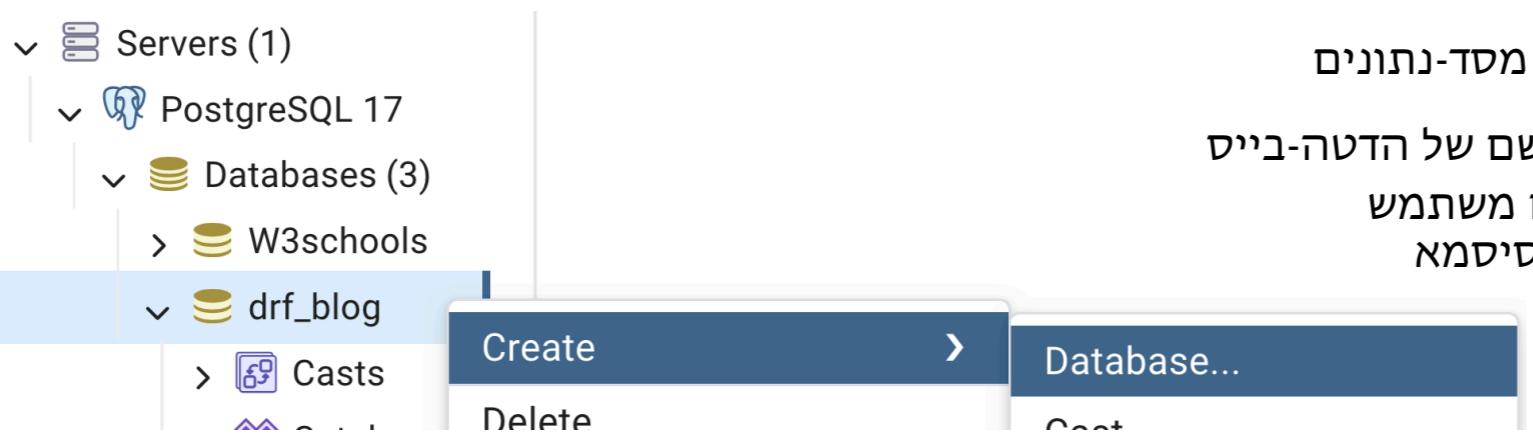
settings.py

```
# show the browsable api:  
REST_FRAMEWORK = {  
    'DEFAULT_RENDERER_CLASSES': [  
        'rest_framework.renderers.JSONRenderer',  
        'rest_framework.renderers.BrowsableAPIRenderer',  
    ]  
}  
  
# hide the browsable api:  
REST_FRAMEWORK = {  
    'DEFAULT_RENDERER_CLASSES': (  
        'rest_framework.renderers.JSONRenderer',  
    )  
}
```

הציגות של Browsable Development מושם בסביבת

הסירה של Browsable Production נבטל בסביבת

עובדת עם מסד נתונים חיצוני:



עובדת עם מסד נתונים חיצוני:

```
export PATH="/Library/PostgreSQL/17/bin:$PATH"
```

```
pip install psycopg2
```

התקנת adapter

2

settings.py

הגדרות

3

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql',  
        'NAME': 'drf_blog',  
        'USER': 'postgres',  
        'PASSWORD': '123456',  
        'HOST': 'localhost',  
        'PORT': '5432',  
    }  
}
```

```
python manage.py migrate
```

4

עובדת עם מסד נתונים חיצוני:

```
python manage.py createsuperuser
```

5

הוספה נתונים במשק ניהול:

6

<http://127.0.0.1:8000/admin>

עבודה עם הספרייה taggit

https://django-taggit.readthedocs.io/en/latest/getting_started.html

<https://django-taggit.readthedocs.io/en/latest/serializers.html>

Usage With Django Rest Framework

Because the tags in *django-taggit* need to be added into a *TaggableManager()* we cannot use the usual *Serializer* that we get from Django REST Framework. Because this is trying to save the tags into a *list*, which will throw an exception.

To accept tags through a *REST* API call we need to add the following to our *Serializer*:

```
from taggit.serializers import (TagListSerializerField,  
                                TaggitSerializer)  
  
class YourSerializer(TaggitSerializer, serializers.ModelSerializer):  
    tags = TagListSerializerField()  
  
    class Meta:  
        model = YourModel  
        fields = '__all__'
```

And you're done, so now you can add tags to your model.

עבודה עם הספרייה taggit

עמוד Posts מראה רק Posts בלי להציג יחס גומלין

serializers.py

```
from .models import Post, Comment, UserProfile, PostUserLikes
from rest_framework.serializers import ModelSerializer
from taggit.serializers import (TagListSerializerField,
                                 TaggitSerializer)

# show tags in posts:
class PostSerializer(TaggitSerializer, ModelSerializer):

    tags = TagListSerializerField()

    class Meta:
        model = Post
        fields = '__all__'
```

כדי להציג מידע מטבלה אחרת נווסף את השדה של tags

עובדת עם הספרייה taggit

Getting Started

To get started using django-taggit simply install it with pip:

```
$ pip install django-taggit
```

Add "taggit" to your project's INSTALLED_APPS setting.

Run `./manage.py migrate`.

And then to any model you want tagging on do the following:

```
from django.db import models

from taggit.managers import TaggableManager

class Food(models.Model):
    # ... fields here

    tags = TaggableManager()
```

Note:

If you want django-taggit to be **CASE-INSENSITIVE** when looking up existing tags, you'll have to set TAGGIT_CASE_INSENSITIVE (in `settings.py` or wherever you have your Django settings) to True (False by default):

```
TAGGIT_CASE_INSENSITIVE = True
```

Settings

The following Django-level settings affect the behavior of the library

`settings.py`

- `TAGGIT_CASE_INSENSITIVE`

```
TAGGIT_CASE_INSENSITIVE = True
```

עבודה עם הספרייה taggit

HTTP <http://127.0.0.1:8000/api/v1/posts/>

POST <http://127.0.0.1:8000/api/v1/posts/>

Params Authorization Headers (8) Body ● Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON** ▾

```
1 {  
2     "tags": [  
3         "API",  
4         "Python"  
5     ],  
6     "title": "Title for Post 3",  
7     "text": "Text for Post 3",  
8     "status": "draft",  
9     "author": 1  
10 }
```

בדיקה:
שליחת בקשה POST תקינה
מייצרת Post וגם שומרת Tags חדשים בדטה-בייס
עם קישור ליחסים גומליים.

שימוש בספרייה
מפלט בשימורה של מידע בטבלת Tags

<http://127.0.0.1:8000/api/v1/posts>

התאמת התצוגה של API

- (1) זה לא נראה טוב
(2) הערך מעובד למחרוזת בודדת

Tags	<pre>[{"name": "REST"}]</pre>
Title	Title for Post 1
Text	Text for Post 1
Status	Draft
Author	itomers

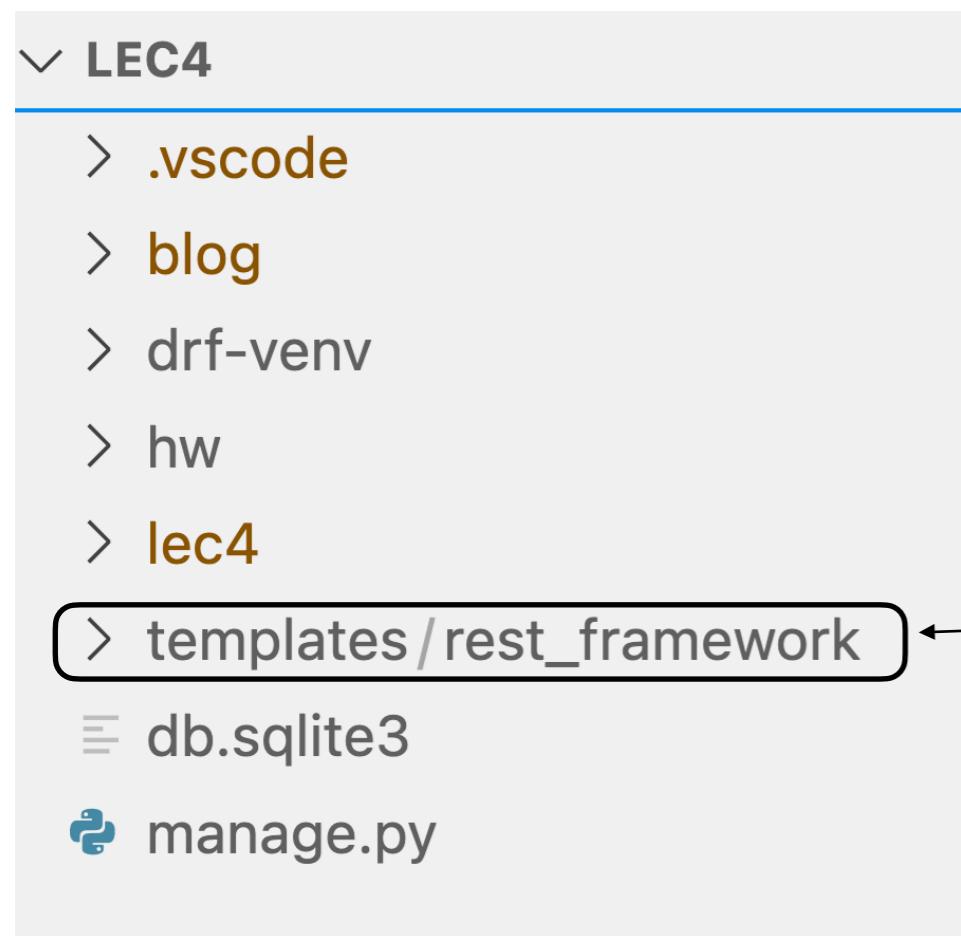
התאמת התצוגה של API

נוריד מכאן את קוד המקור של browsable API

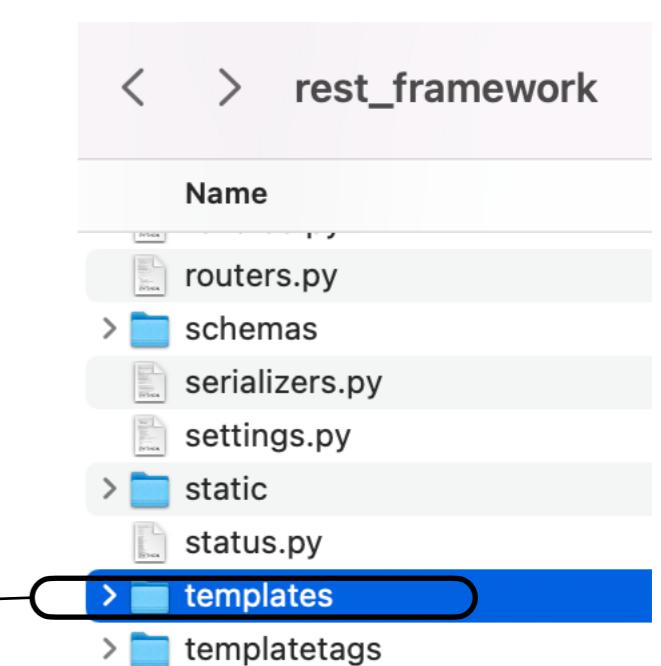
<https://github.com/encode/django-rest-framework/tree/master>

התקינה templates
מיכלה קבצי html שאוטם נרצה לעירוף

התקינה שהורדנו מיכיל rest_framework



נשים לב לכלול את התקינה
שהורדנו בתיקיה הראשית
מחוץ לכל התקיות



התאמת הציגות של API

הגדרות לשימוש בתבניות שהורדנו

settings.py

```
# a file/dir module
import os

TEMPLATES = [
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [os.path.join(BASE_DIR, 'templates')],
    'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
            'django.contrib.auth.context_processors.auth',
            'django.contrib.messages.context_processors.messages',
        ],
    },
},
]
```

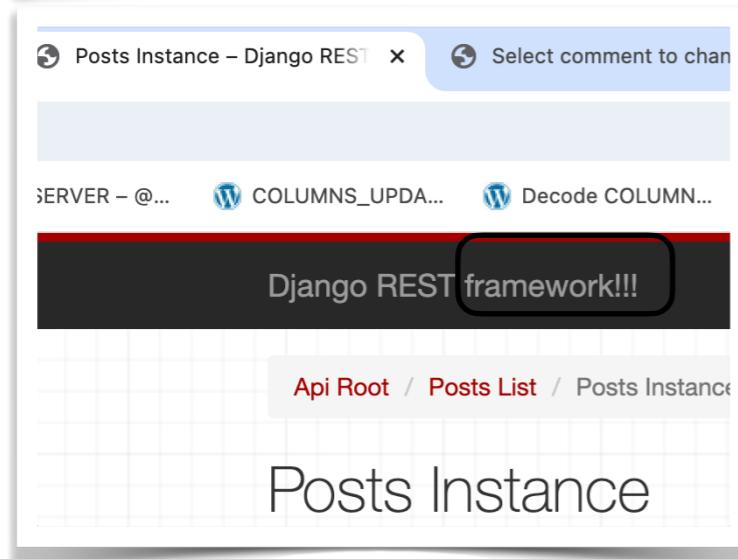
התאמת התצוגה של browsable API

templates/rest_framework/base.html



A screenshot of a code editor showing the `base.html` template. The code includes a `` block with a `{% block branding %}` placeholder and an `<a>` tag with the text "Django REST framework!!!". A blue rounded rectangle highlights the word "framework!!!".

```
<span>
    {% block branding %}
        <a class='navbar-brand' rel="nofollow" href="{% url 'api-root' %}>
            Django REST framework!!!
        </a>
    {% endblock %}
</span>
```



התאמת הציגות של API

filtrim b Django Templates

מקבל קלט-> מוציא פלט מעובד

אצלנו בפרויקט-tagiot זה רשימה של מחוזות



הפלט ייצור מחוזת אחת שמיופרדת ברווחים

יצירת פילטר - Custom Template Tag

<https://docs.djangoproject.com/en/5.1/howto/custom-template-tags>

```
from django import template
```

```
register = template.Library()
```

```
def lower(value): # Only one argument.  
    """Converts a string into all lowercase"""\n    return value.lower()
```

```
@register.filter(name="cut")\ndef cut(value, arg):\n    return value.replace(arg, "")
```

יצירת פילטר - Custom Template Tag

<https://docs.djangoproject.com/en/5.1/howto/custom-template-tags/>

ניצור תיוקה וקובץ:

1

LEC4

- > .vscode
- > blog
- > drf-venv
- > hw
- > lec4
- > templates
- < templates
 custom_filters.py

≡ db.sqlite3

⌘ manage.py

2

לעיבוד מחרוזות

```
from django import template
register = template.Library()

@register.filter
def join_space(value):
    if isinstance(value, (list, tuple)):
        return " ".join(value)
    return value

# usage:
# include templatetags in settings.py
# {% load custom_filters %}
# {% tags|join_space %}
```

יצירת פילטר - Custom Template Tag

<> `textarea.html` U X

templates > rest_framework > horizontal > <> `textarea.html`

1 `{% load custom_filters %}`

`{{ field.value|join_space }}`

יצירת פילטר - Custom Template Tag

```
import os
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            os.path.join(BASE_DIR, 'templates'),
            os.path.join(BASE_DIR, 'templatetags')
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Custom Serializer Field:

<https://www.django-rest-framework.org/api-guide/fields/#a-basic-custom-field>

הלקוח שולח מידע בפורמט של מחרוזת עם רווחים
אנחנו רוצים לשמר את זה בדטה-בייס כרשימה של מחרוזות

- התפקיד של Serializer
- (1) להמיר אובייקט לJSON
- (2) להמיר JSON לאובייקט

```
class Color:  
    """  
    A color represented in the RGB colorspace.  
    """  
  
    def __init__(self, red, green, blue):  
        assert(red >= 0 and green >= 0 and blue >= 0)  
        assert(red < 256 and green < 256 and blue < 256)  
        self.red, self.green, self.blue = red, green, blue  
  
class ColorField(serializers.Field):  
    """  
    Color objects are serialized into 'rgb(#, #, #)' notation.  
    """  
  
    def to_representation(self, value):  
        return "rgb(%d, %d, %d)" % (value.red, value.green, value.blue)  
  
    def to_internal_value(self, data):  
        data = data.strip('rgb(').rstrip(')')  
        red, green, blue = [int(col) for col in data.split(',')]  
        return Color(red, green, blue)
```

המידע מגיע כמחרוזת עם
רווחים
או עם פסיקים

ואנחנו רוצים להמיר אותו
לרשימה של מחרוזות

שינוי התנהגות לפי סוג הלקוח:

<https://www.djangoproject.com/en/2.0/topics/http/middleware/>

נרצה לדעת מהו ה乞עה מTİ:הבקשה מגיעה:

3 אפשרויות:

html, api, json

Varying behavior by media type

In some cases you might want your view to use different serialization styles depending on the accepted media type. If you need to do this you can access `request.accepted_renderer` to determine the negotiated renderer that will be used for the response.

For example:

```
@api_view(['GET'])
@renderer_classes([TemplateHTMLRenderer, JSONRenderer])
def list_users(request):
    """
    A view that can return JSON or HTML representations
    of the users in the system.
    """
    queryset = Users.objects.filter(active=True)

    if request.accepted_renderer.format == 'html':
        # TemplateHTMLRenderer takes a context dict,
        # and additionally requires a 'template_name'.
        # It does not require serialization.
        data = {'users': queryset}
        return Response(data, template_name='list_users.html')

    # JSONRenderer requires serialized data as normal.
    serializer = UserSerializer(instance=queryset)
    data = serializer.data
    return Response(data)
```

שינוי התחנכות לפי סוג הלקוח:

serializers.py

הרעיוון הכלומי לתקן התחנכות:

```
class TagField(TagListSerializerField):

    def to_internal_value(self, value):
        request = self.context.get('request')

        # if browsable api:
        if request and request.accepted_renderer.format == 'api':
            # adjust the value to match our desired db format
            value = value[0].split(" ")

        return super().to_internal_value(value)
```

שינוי התנהגות לפי סוג הלקוח:

עוד בדיקות - שהבקשה קיימת ומכילה accepted_renderer

עוד בדיקות - שמה שקיבלנו זהה בדיק רשימה עם אלמנט אחד מסוג מחרוזת:

serializers.py

```
from taggit.serializers import (TagListSerializerField,
                                 TaggitSerializer)

class TagField(TagListSerializerField):

    def to_internal_value(self, value):
        request = self.context.get('request')

        is_browsable_api = (
            request
            and hasattr(request, 'accepted_renderer')
            and request.accepted_renderer.format == 'api'
        )

        if (
            is_browsable_api
            and isinstance(value, list)
            and len(value) == 1
            and isinstance(value[0], str)
        ):
            value = value[0].split() #by default splits by space

        return super().to_internal_value(value)
```

שינוי התנהלות לפי סוג הלקוח:

שימוש ב-Custom Field

serializers.py

```
class PostSerializer(TaggitSerializer, ModelSerializer):
    tags = TagField(style={'base_template': 'textarea.html'})  
  
class Meta:  
    model = Post  
    fields = '__all__'
```

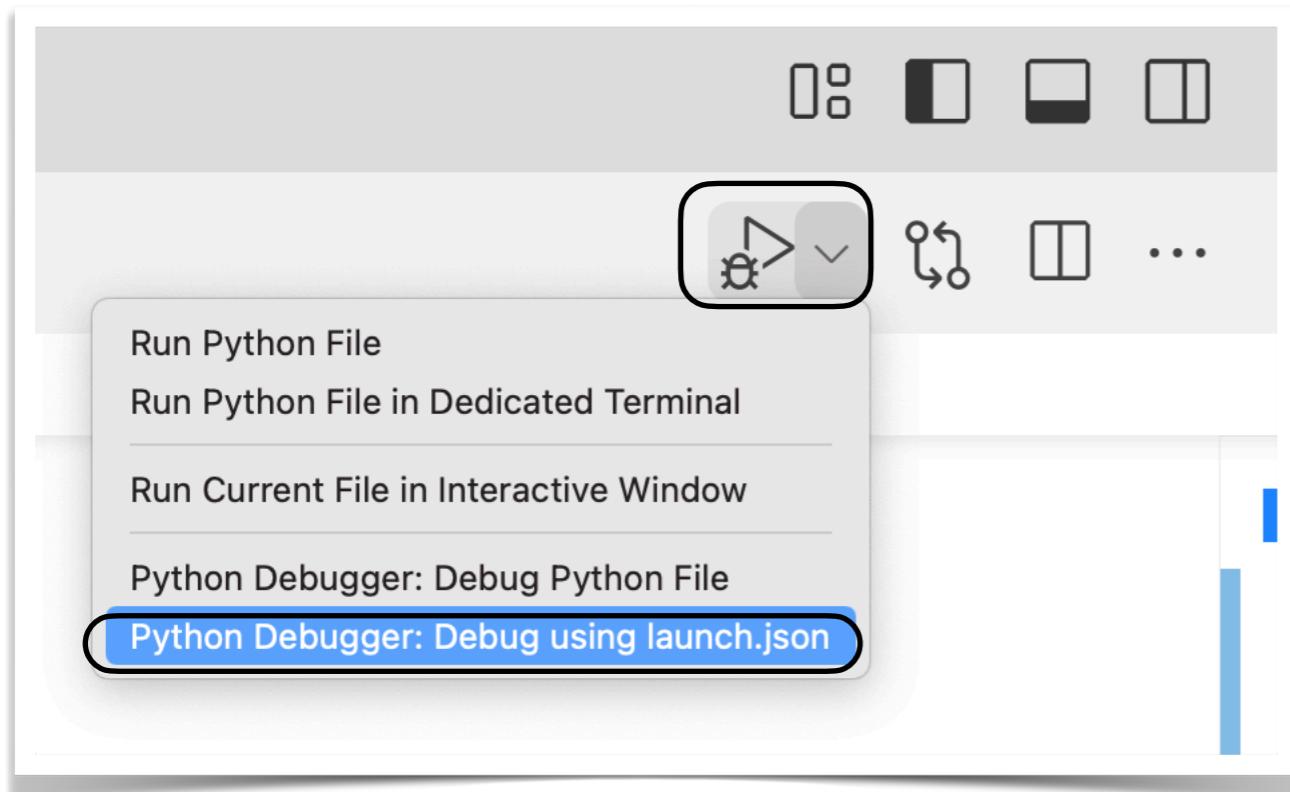
לכל שדה שמגדירים בSerializer
ניתן לקבוע באיזו תבנית להשתמש
מתוך התבניות בתיקיה templates

אפשר לדוגמא input.html

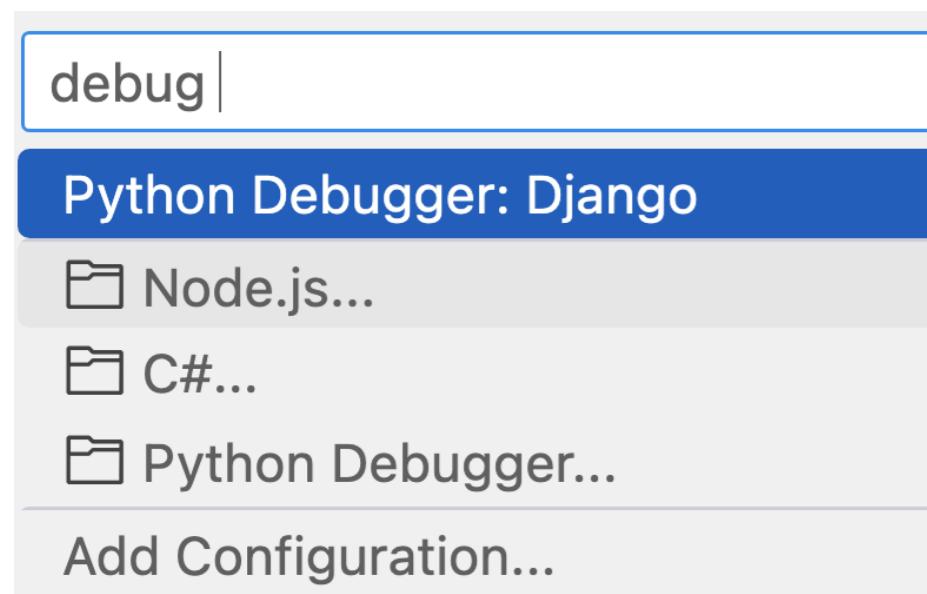
Debugging :-)

```
7 class TagField(TagListSerializerField):  
8  
9     def to_internal_value(self, value):  
10        request = self.context.get('request')  
11  
12        is_browsable_api = (  
13            request  
14            and hasattr(request, 'accepted_renderer')  
15            and request.accepted_renderer.format == 'api'  
16        )
```

Debugging :-)



Python Debugger



Debugging :-)

Code File Edit Selection View Go Run Terminal Window Help

RUN AND DEBUG Python Debugger: D ...

VARIABLES

Locals

- > special variables
- > self = TagField(style={'base_template': 'tex...', 'value': ['abc']}
- > Globals

blog > serializers.py > TagField > to_internal_value

```
class TagField(TagListSerializerField):  
    def to_internal_value(self, value): self = TagField(style={  
        request = self.context.get('request')
```

step over

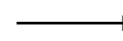
התקדם לשורה הבאה

continue

המשך את הדיבאגר

הציגת התגובה:

```
[  
  {  
    "id": 1,  
    "text": "Nice Post",  
    "created_at": "2025-02-23T08:56:04.937995Z",  
    "updated_at": "2025-02-23T08:56:04.938018Z",  
    "author": 3,  
    "post": 1,  
    "reply_to": null  
  },  
  {  
    "id": 2,  
    "text": "Thanks Jhon",  
    "created_at": "2025-02-23T08:56:20.606355Z",  
    "updated_at": "2025-02-23T08:56:20.606369Z",  
    "author": 2,  
    "post": 1,  
    "reply_to": 1  
  },  
  {  
    "id": 3,  
    "text": "Great post",  
    "created_at": "2025-02-23T08:56:41.592412Z",  
    "updated_at": "2025-02-23T08:56:41.592425Z",  
    "author": 3,  
    "post": 2,  
    "reply_to": null  
  },  
  {  
    "id": 4,  
    "text": "you welcome",  
    "created_at": "2025-02-23T08:57:08.155769Z",  
    "updated_at": "2025-02-23T08:57:08.155830Z",  
    "author": 3,  
    "post": 1,  
    "reply_to": 2  
  }]
```



```
[  
  {  
    "id": 1,  
    "text": "Nice Post",  
    "created_at": "2025-02-23T08:56:04.937995Z",  
    "updated_at": "2025-02-23T08:56:04.938018Z",  
    "author": 3,  
    "post": 1,  
    "reply_to": null,  
    "replies": [  
      {  
        "id": 2,  
        "text": "Thanks Jhon",  
        "created_at": "2025-02-23T08:56:20.606355Z",  
        "updated_at": "2025-02-23T08:56:20.606369Z",  
        "author": 2,  
        "post": 1,  
        "reply_to": 1,  
        "replies": [  
          {  
            "id": 4,  
            "text": "you welcome",  
            "created_at": "2025-02-23T08:57:08.155769Z",  
            "updated_at": "2025-02-23T08:57:08.155830Z",  
            "author": 3,  
            "post": 1,  
            "reply_to": 2  
          }  
        ]  
      }  
    ]  
  },  
  {  
    "id": 3,  
    "text": "Great post",  
    "created_at": "2025-02-23T08:56:41.592412Z",  
    "updated_at": "2025-02-23T08:56:41.592425Z",  
    "author": 3,  
    "post": 2,  
    "reply_to": null  
  }]
```

הציגת התשובות:

list->dictionary

```
#1) list of comments
comments = res.data

#2) dictionary of comments
comments_dict = {comment["id"]:comment for comment in comments}

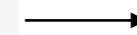
#3) list of root comments
root_comments = []
```



לכאן נכנס את התוצאות אחרי העיבוד

הציגת התגובהות:

```
{  
    "id": 1,  
    "text": "Nice Post",  
    "created_at": "2025-02-23T08:56:04.937995Z",  
    "updated_at": "2025-02-23T08:56:04.938018Z",  
    "author": 3,  
    "post": 1,  
    "reply_to": null  
},
```



נרצה להוסיף []
אבל רק להודעה שיש לה reply

אם ההודעה היא הודעת "אב" של הודעה אחרת - נוסיף ל"אב" שדה רשימה ריקה בשם replies

```
if parent:  
    # parent has no replies:  
    if "replies" not in parent:  
        parent["replies"] = []
```

הציגת התגובהות:

```
root_comments = []

# loop over the list:
for comment in comments:
    # extract the parent:
    parent_id = comment['reply_to']
    # if root comment:
    if parent_id is None:
        root_comments.append(comment)
    else:
        parent = comments_dict.get(parent_id)
        if parent:
            # parent has no replies:
            if "replies" not in parent:
                parent["replies"] = []
            parent["replies"].append(comment)

res.data = root_comments
```

זכור שאובייקטים מנוהלים לפי מצביע בזיכרון
עריכה של אובייקט במקום אחד
משפיעה על אותו אובייקט בזיכרון באופן גלובלי

הציגת התגובה:

```
class CommentsViewSet(ModelViewSet):
    queryset = Comment.objects.all()
    serializer_class = CommentSerializer

    def list(self, request, *args, **kwargs):
        res = super().list(request, *args, **kwargs)

        #1) list of comments
        comments = res.data

        #2) dictionary of comments
        comments_dict = {comment["id"]:comment for comment in comments}

        #3) list of root comments
        root_comments = []

        # loop over the list:
        for comment in comments:
            # extract the parent:
            parent_id = comment['reply_to']
            # if root comment:
            if parent_id is None:
                root_comments.append(comment)
            else:
                parent = comments_dict.get(parent_id)
                if parent:
                    # parent has no replies:
                    if "replies" not in parent:
                        parent["replies"] = []
                    parent["replies"].append(comment)

        res.data = root_comments
        return res
```

זכור שאובייקטים מנוהלים לפי מצביע בזיכרון
עריכה של אובייקט במקום אחד
משפיעה על אותו אובייקט בזיכרון באופן גלובלי

תגובה לתגובה חייבת עם אותו ID

```
class CommentsViewSet(ModelViewSet):
    queryset = Comment.objects.all()
    serializer_class = CommentSerializer

    def create(self, request, *args, **kwargs):
        data = request.data
        reply_to = data.get('reply_to')
        post_id = data.get('post')

        if reply_to:
            replied = Comment.objects.get(id=reply_to)
            print(replied)
            if (
                replied and replied.post.id != post_id
            ):
                return Response(
                    {"error": "Reply must be on the same post"},
                    status=400
                )

        return super().create(request, *args, **kwargs)
```