

# Django

TomerBu

# **נושאים להיום:**

Django Rest Framework

מבוא

התקנה והגדרות

Request Response Classes

Serializers

Function Based Views

# שיעור בית:

צרו את המודלים הבאים:

1

Category:  
name

שם הקטגוריה - טקסט קצר

קטגוריות דוגמא:

תבניות, פיתון, LOLAOT, HTTP, וכו'...

2

Post:  
title  
content  
created  
modified  
categories

קשר של רבים ל רבים  
ManyToManyField

3

Author:  
birthday  
user

קשר של אחד לאחד

4

Comment:  
author  
text  
created  
post

קשר של אחד ל רבים:  
models.ForeignKey

קשר של אחד ל רבים:  
models.ForeignKey

זרעו את המודלים במאזן הנתונים

שים לב ליצור ולידציות

# שיעור בית:

```
django-admin startproject lec3
```

```
vscode ג lec3 נפתח את התיקייה
```

```
python3 -m venv .venv
```

חלוןנות

```
.venv/bin/activate
```

מאק

```
source .venv/bin/activate
```

התקנות

```
pip install django
```

יצירת app

```
python manage.py startapp hw → INSTALLED_APPS += ['hw']
```

# שיעור בית:

```
from django.db import models

class Category(models.Model):
    name = models.CharField(max_length=200)

    def __str__(self):
        return self.name
```

```
class Post(models.Model):
    title = models.CharField(max_length=200, unique=True, blank=False, null=False)
    content = models.TextField(blank=False, null=False)
    categories = models.ManyToManyField(Category, related_name='posts')
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return f'{self.title} - {self.id}'
```

```
from django.contrib.auth.models import User

class Author(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    birthday = models.DateField()

    def __str__(self):
        return self.user.username
```

# שיעור בית:

```
class Comment(models.Model):
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
    post = models.ForeignKey(Post, on_delete=models.CASCADE)
    text = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

```
python manage.py makemigrations hw
python manage.py migrate
```

```
from django.contrib import admin

# Register your models here.

from .models import Post, Author, Category, Comment

admin.site.register([Post, Author, Category, Comment])

# python manage.py createsuperuser
```

# שיעור בית:



hw/management/commands/seed\_db

```
from django.core.management.base import BaseCommand

class Command(BaseCommand):
    help = 'Seeds the database with initial data'

    def handle(self, *args, **kwargs):
        self.stdout.write(self.style.SUCCESS('Seeding the database...'))
```

**python manage.py seed\_db**

# שיעור בית:

```
from django.core.management.base import BaseCommand
from django.contrib.auth.models import User
from hw.models import *

class Command(BaseCommand):
    help = 'Seeds the database with initial data'

    def handle(self, *args, **kwargs):
        self.stdout.write(self.style.SUCCESS('Seeding the database...'))

        user, created = (User.objects.
                          get_or_create(
                              username='admin',
                              defaults={
                                  'is_superuser': True, 'is_staff': True
                              }))
        user.set_password('123456') ← הצפת הסיסמה
        user.save() ← שמירה נוספת

        self.stdout.write(self.style.SUCCESS('Database seeded'))
```

הדפסה בצבע

הצפת הסיסמה ←

שמירה נוספת ←

# שיעור בית:

```
python manage.py makemigration hw --empty --name seed
```

```
from django.db import migrations
from django.core.management import call_command

class Migration(migrations.Migration):

    def run_seed(apps, schema_editor):
        call_command('seed_db')

    dependencies = [
        ('hw', '0001_initial'),
    ]

    operations = [
        migrations.RunPython(run_seed),
    ]

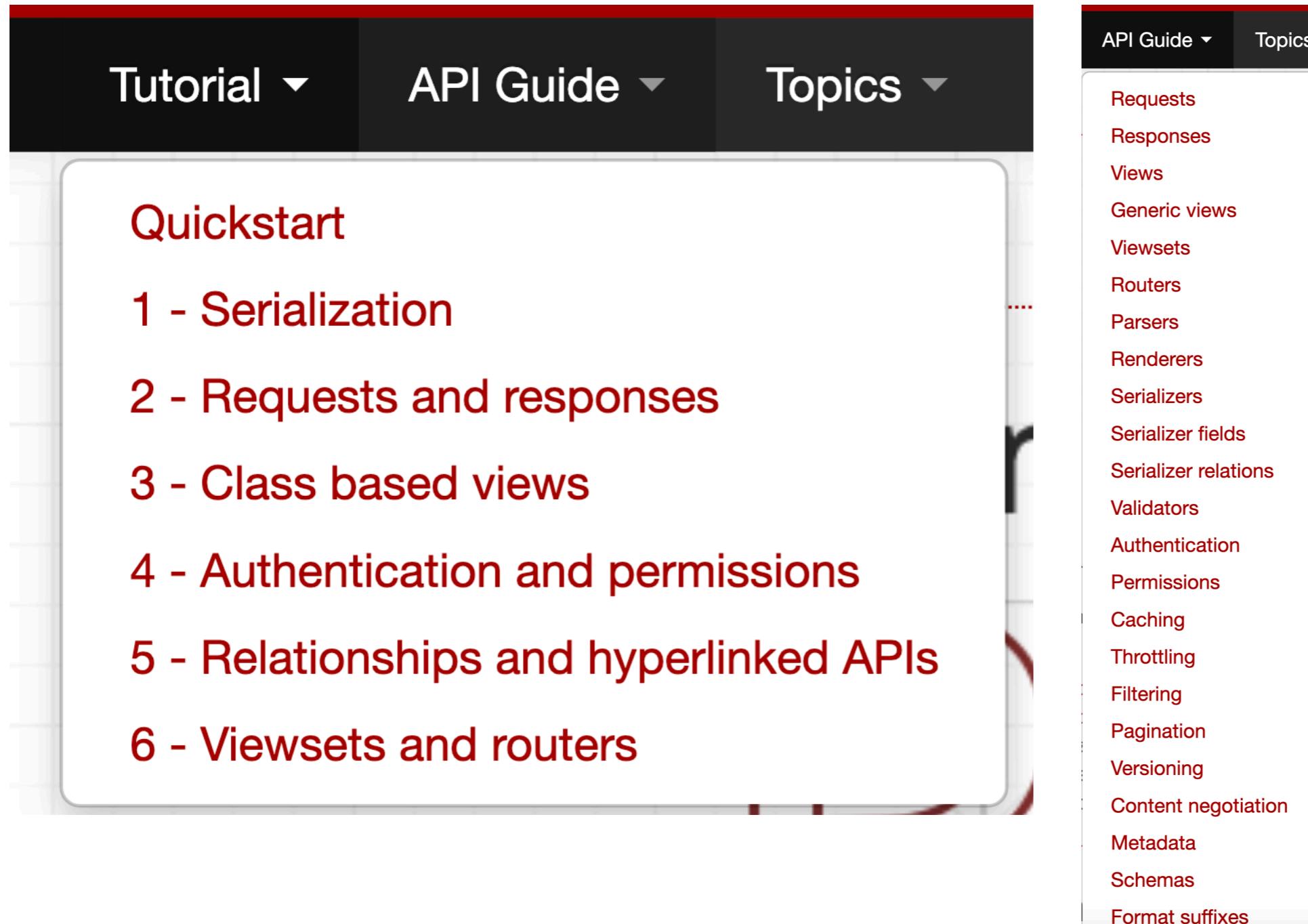
# python manage.py migrate hw
```

```
python manage.py migrate hw
```

```
python manage.py runserver
```

# django-rest-framework

<https://www.djangoproject.com/>



The screenshot shows the official documentation for Django REST Framework. At the top, there is a navigation bar with three main items: "Tutorial" (with a dropdown arrow), "API Guide" (with a dropdown arrow), and "Topics" (with a dropdown arrow). The "API Guide" item is currently selected, indicated by a red background. Below the navigation bar, the page content begins with a section titled "Quickstart". This section contains a numbered list from 1 to 6, each representing a different topic: "1 - Serialization", "2 - Requests and responses", "3 - Class based views", "4 - Authentication and permissions", "5 - Relationships and hyperlinked APIs", and "6 - Viewsets and routers". To the right of the "Quickstart" section, there is a vertical sidebar containing a list of topics under the heading "API Guide". The topics listed are: Requests, Responses, Views, Generic views, Viewsets, Routers, Parsers, Renderers, Serializers, Serializer fields, Serializer relations, Validators, Authentication, Permissions, Caching, Throttling, Filtering, Pagination, Versioning, Content negotiation, Metadata, Schemas, and Format suffixes.

API Guide Topics

- Requests
- Responses
- Views
- Generic views
- Viewsets
- Routers
- Parsers
- Renderers
- Serializers
- Serializer fields
- Serializer relations
- Validators
- Authentication
- Permissions
- Caching
- Throttling
- Filtering
- Pagination
- Versioning
- Content negotiation
- Metadata
- Schemas
- Format suffixes

# django-rest-framework

## התקנה:

```
pip install djangorestframework
```

Add '`rest_framework`' to your `INSTALLED_APPS` setting.

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

```
python manage.py startapp api
```

# django-rest-framework

## התקנה:

api/views.py

```
from django.shortcuts import render
from django.http import HttpResponse

def index(request):
    return HttpResponse("ok")
```

python manage.py runserver

lec3/api/urls.py

```
from django.urls import path
from .views import index

urlpatterns = [
    path('students/', index, name='students'),
]
```

नियर קובץ urls

בדיקה:

<http://localhost:8000/api/students>

lec3/urls.py

נערוך את קובץ urls של lec3

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

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

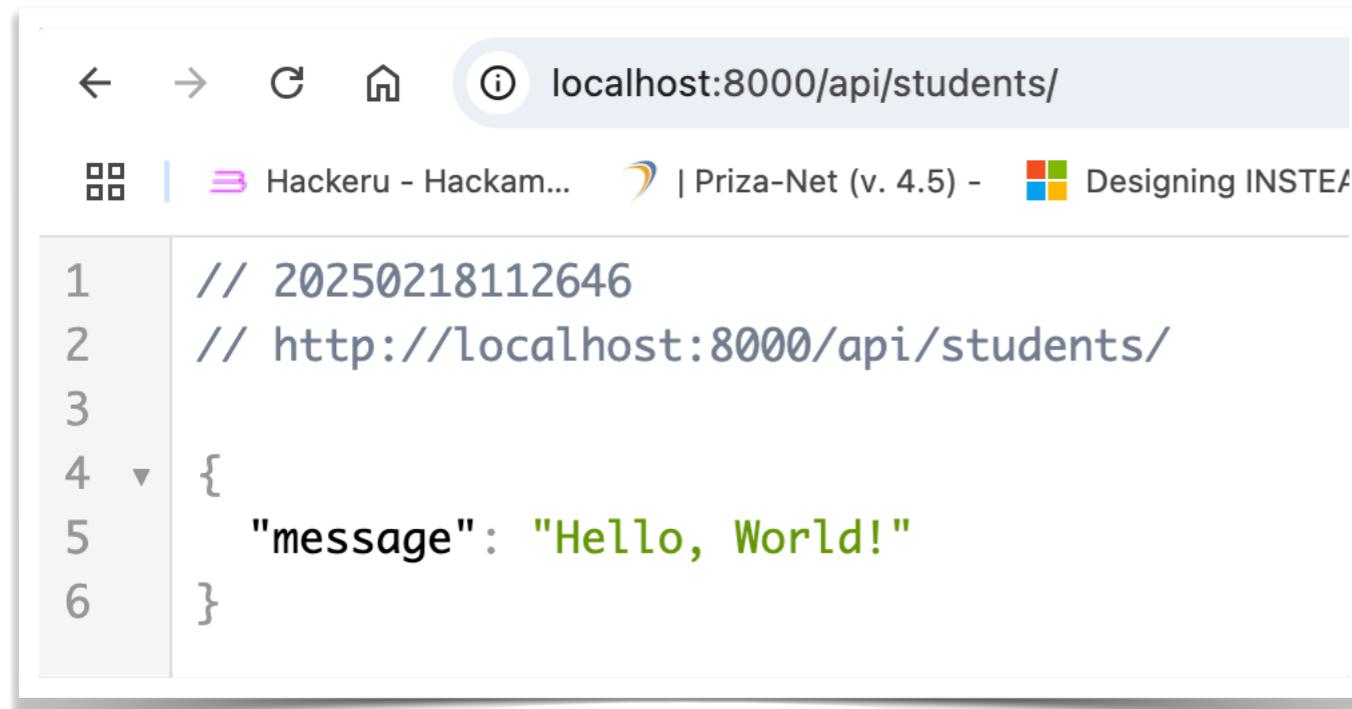
# django-rest-framework

## מבוא לצורך:

api/views.py

```
from django.shortcuts import render
from django.http import HttpResponseRedirect, JsonResponse

# return JsonResponse
def index(request):
    return JsonResponse({'message': 'Hello, World!'})
```



# django-rest-framework

## מבוא לצורך:

```
from django.http import JsonResponse
from .models import Student

# return JsonResponse
def index(request):
    students = Student.objects.all()
    return JsonResponse({'message': students})
```

### TypeError at /api/students/

Object of type QuerySet is not JSON serializable

Request Method: GET

Request URL: http://localhost:8000/api/students/

Django Version: 5.1.6

Exception Type: TypeError

Exception Value: Object of type QuerySet is not JSON serializable

Exception Location: /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/json/encoder.py, line 180, in default

Raised during: api.views.index

Python Executable: /Volumes/E4TB/W150824MR/Django/lec3/.venv/bin/python

Python Version: 3.13.1

Python Path: ['/Volumes/E4TB/W150824MR/Django/lec3',
 '/Library/Frameworks/Python.framework/Versions/3.13/lib/python313.zip',
 '/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13',
 '/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/lib-dynload',
 '/Volumes/E4TB/W150824MR/Django/lec3/.venv/lib/python3.13/site-packages']

Server time: Tue, 18 Feb 2025 09:37:39 +0000

# פורמט להעברת נתונים:

פורמט להעברת נתונים בין השפות השונות:

בקשת HTTP נשלח JSON או XML

```
1 <song position="6">
2   <artist>65daysofstatic</artist>
3   <title>Come to Me</title>
4   <album>We Were Exploding Anyway</album>
5   <release>26 April 2010</release>
6   <duration>8:00</duration>
7   <special>Robert Smith, The Cure</special>
8 </song>
```

סריליזציה:

תרגום של אובייקט משפט תכנות לפורמט רצוי (דוגמא XML/JSON)

דה-סריליזציה:

פונוח של JSON/XML ויצירת אובייקט בשפת התכנות

# Serialization and De-Serialization:

מחלקה של :Student  
אובייקט מהמחלקה Student

הוא אובייקט בפייתון (הוא לא Json)

**Serialization:**  
המרת אובייקט משפת תכנות לJSON או XML

**De-Serialization:**  
המרת JSON לאובייקט בשפת תכנות

# Serialization and De-Serialization:

serializers.py

נגידır Serializer שמודע לתרגם אובייקט לJSON

```
from rest_framework import serializers

# serializers convert objects to json and vice versa

class StudentSerializer(serializers.Serializer):
    name = serializers.CharField(max_length=100)
```

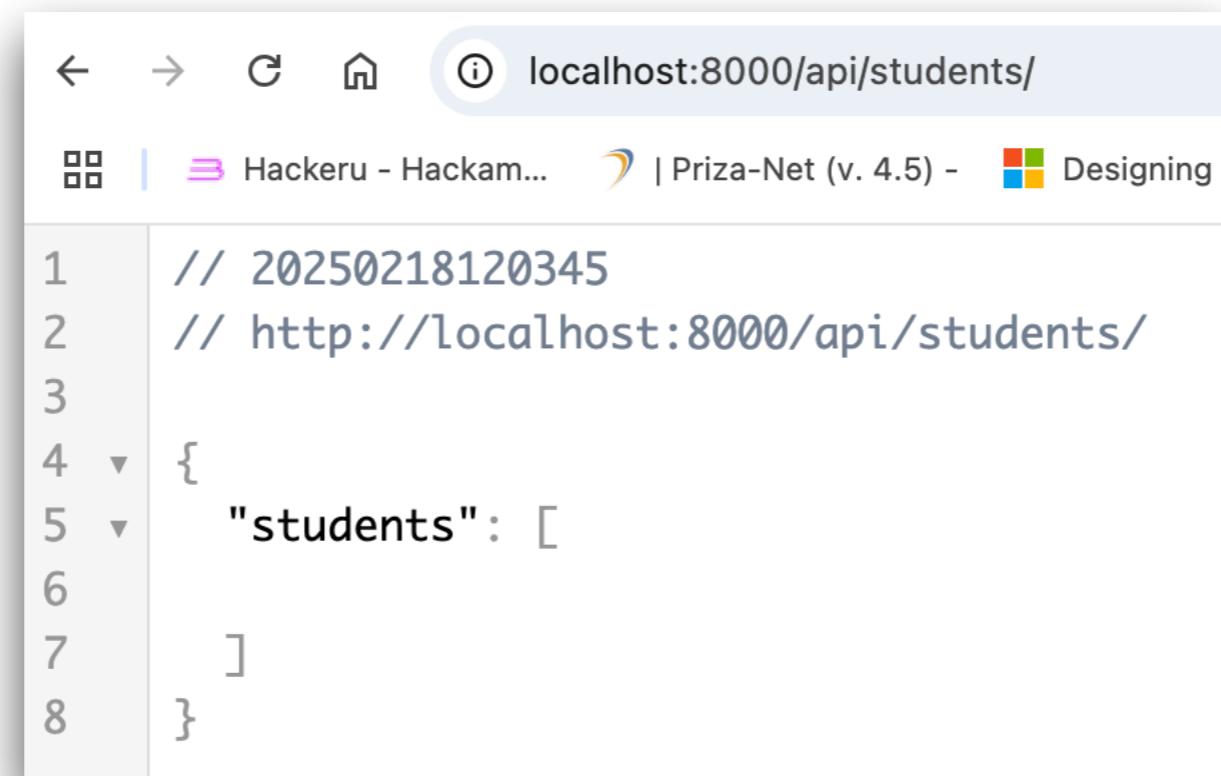
```
from django.http import JsonResponse
from .models import Student
from .serializers import StudentSerializer

# return JsonResponse
def index(request):
    students = Student.objects.all()
    serializer = StudentSerializer(students, many=True)
    return JsonResponse({'students': serializer.data})
```

יצירת מופע של ה-StudentSerializer

JSON

# Serialization and De-Serialization:



A screenshot of a web browser window displaying a JSON response. The address bar shows the URL `localhost:8000/api/students/`. The page content is a JSON object with the following structure:

```
1 // 20250218120345
2 // http://localhost:8000/api/students/
3
4 {
5   "students": [
6
7   ]
8 }
```

# Serialization and De-Serialization:

Serializer יכול לקבל מידע מבקשת של POST  
לטובת ייצור מופע חדש - לשם כך קיימת המתודה ()  
create()

Serializer יכול לקבל מידע מבקשת של PUT  
לטובת ייצור עדכון מופע קיים - לשם כך קיימת המתודה ()  
update()

```
from rest_framework import serializers

# serializers convert objects to json and vice versa
from .models import Student
class StudentSerializer(serializers.Serializer):
    name = serializers.CharField(max_length=100)

    def update(self, instance, validated_data):
        # copy all the props from the request to db object
        instance.name = validated_data['name']

        # save the db object
        instance.save()
        return instance

    def create(self, validated_data):
        # Student.objects.create(name=validated_data['name'])
        return Student.objects.create(**validated_data)
```

# Serialization and De-Serialization:

Serializer יכול לקבל מידע מבקשת של POST  
לטובת ייצור מופע חדש - לשם כך קיימת המתודה ()  
create()

Serializer יכול לקבל מידע מבקשת של PUT  
לטובת ייצור עדכון מופע קיים - לשם כך קיימת המתודה ()  
update()

```
from rest_framework import serializers

# serializers convert objects to json and vice versa
from .models import Student
class StudentSerializer(serializers.Serializer):
    name = serializers.CharField(max_length=100)

    def update(self, instance, validated_data):
        # copy all the props from the request to db object
        instance.name = validated_data.get('name', instance.name)

        # save the db object
        instance.save()
        return instance

    def create(self, validated_data):
        # Student.objects.create(name=validated_data['name'])
        return Student.objects.create(**validated_data)
```

# Request/Response Objects

סורקת את ה View שלנו  
ומייצרת עבורנו דוקומנטציה:

כדי שזה יעבד:  
יש לוודא שמחזירים Response  
ולא העיפושים של Django

```
from rest_framework.response import Response
from rest_framework.request import Request
from rest_framework.decorators import api_view
from .models import Student
from .serializers import StudentSerializer

@api_view(['GET', 'POST'])
def index(request:Request):
    students = Student.objects.all()
    serializer = StudentSerializer(students, many=True)
    return Response({'students': serializer.data})
```

# Browsable API:

Django REST framework

itomers

Index

Index      OPTIONS      GET ▾

GET /api/students/

HTTP 200 OK  
Allow: OPTIONS, GET, POST  
Content-Type: application/json  
Vary: Accept

```
{  
    "students": []  
}
```

Media type: application/json

Content:

```
{  
|  
}
```

POST

The screenshot shows a browsable API interface for a Django REST framework application. At the top, there's a dark header bar with the text 'Django REST framework' and 'itomers'. Below it, a 'Index' section is shown. On the left is the word 'Index'. To its right are two blue buttons: 'OPTIONS' and 'GET' with a dropdown arrow. Underneath is a grey box containing a 'GET' method for the endpoint '/api/students/'. The response status is 'HTTP 200 OK' with headers: 'Allow: OPTIONS, GET, POST', 'Content-Type: application/json', and 'Vary: Accept'. The JSON response body is shown as: '{ "students": [] }'. Below this is another section titled 'Media type:' with a dropdown set to 'application/json'. Under 'Content:', there's a large text area containing an empty JSON object: '{ | }'. At the bottom right of this section is a blue 'POST' button.

# התמודדות עם בקשה POST

זה-סרייליזציה:  
מקבל JSON מהלכו ו ממיר לאובייקט

```
@api_view( ['GET', 'POST'])
def index(request:Request):

    if request.method == 'POST':
        # serializer is in create mode:
        serializer = StudentSerializer(data = request.data)
        if serializer.is_valid():
            serializer.save() # will invoke create
            return Response(serializer.data, status=201)
        return Response(serializer.errors, status=400)

    students = Student.objects.all()
    serializer = StudentSerializer(students, many=True)
    return Response({'students': serializer.data})
```

סטוס לבקשת POST מוצלחת

# התמודדות עם בקשת POST

Index

OPTIONS GET ▾

GET /api/students/

HTTP 200 OK  
Allow: POST, OPTIONS, GET  
Content-Type: application/json  
Vary: Accept

```
{  
  "students": [  
    {  
      "name": "Larry"  
    },  
    {  
      "name": "Larry"  
    },  
    {  
      "name": "L"  
    }  
  ]  
}
```

Media type: application/json

Content: {"name": "Larry"}

Make a POST request on Index resource

POST

The screenshot shows a REST API interface. At the top, there are buttons for 'OPTIONS' and 'GET'. Below that, a 'GET /api/students/' request is shown with a 200 OK status. The response body is a JSON array containing three student objects, each with a name field. Below this, there's a form for making a POST request to the same endpoint. The 'Media type:' dropdown is set to 'application/json', and the 'Content' field contains the JSON object {"name": "Larry"}. A tooltip above the 'POST' button says 'Make a POST request on Index resource'. The entire interface has a light gray background with blue highlights for buttons and input fields.

# בדיקות תקינות של נתוניים בבקשת:

```
from rest_framework import serializers

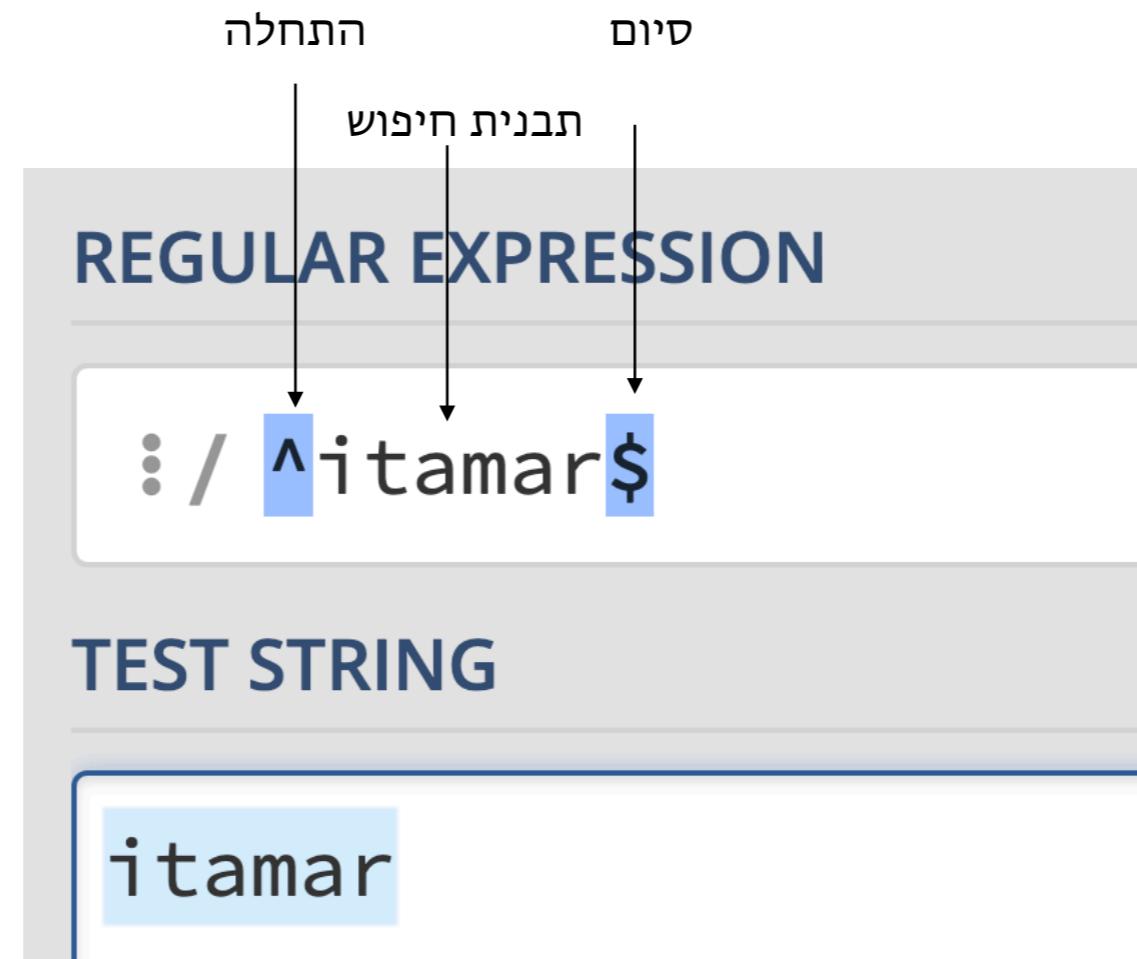
# serializers convert objects to json and vice versa
from .models import Student
from django.core.validators import MinLengthValidator

class StudentSerializer(serializers.Serializer):
    name = serializers.CharField(max_length=100, validators = [
        MinLengthValidator(2)
    ])
    id = serializers.IntegerField(read_only=True)

    def update(self, instance, validated_data):
        instance.name = validated_data.get('name', instance.name)
        instance.save()
        return instance

    def create(self, validated_data):
        return Student.objects.create(**validated_data)
```

# מבוא לביטויים רגולריים:



# מבוא לביטויים רגולריים:

The screenshot shows the regex101.com interface. In the center, under the "REGULAR EXPRESSION" tab, the pattern `^itamar$` is displayed with a green "1 match (9 steps, 85µs)" status bar. Below it, the "TEST STRING" field contains the word `itamar`. To the right, the "EXPLANATION" section provides a breakdown of the regex: `^` asserts position at start of a line, `itamar` matches the characters `itamar` literally (case sensitive), and `$` asserts position at the end of a line. It also mentions global pattern flags and a note about modifiers.

regular expressions 101

SAVE & SHARE

Save new Regex `⌘+s`

Add to Community Libr...

FLAVOR

`</>` PCRE2 (PHP  $\geq 7.3$ ) ✓

`</>` PCRE (PHP <7.3)

REGULAR EXPRESSION 1 match (9 steps, 85µs) `i`

`^itamar$ / gm`

TEST STRING

itamar

EXPLANATION

`^` asserts position at start of a line ?

`itamar` matches the characters `itamar` literally (case sensitive)

`$` asserts position at the end of a line ?

Global pattern flags

`gm` modifier: global All matches (don't return after first match)

# מבוא לביטויים רגולריים:

אותיות אפשריות

כמויות 3 ומעלה

REGULAR EXPRESSION

1 match (4 steps, 55µs) i

`:[ / ^[a-z]{3,} $` / gm 📋

TEST STRING

abc

The screenshot shows a regular expression testing interface. At the top, it says "REGULAR EXPRESSION" and "1 match (4 steps, 55µs)" with an information icon. Below that is the regex pattern: `:[ / ^[a-z]{3,} \$` with the quantifier `{3,}` highlighted in orange. To the right of the pattern are the flags "/gm" and a copy icon. Below the pattern is a "TEST STRING" input field containing "abc". Arrows from the text above point to the start of the regex pattern and the quantifier respectively.

# מבוא לביטויים רגולריים:

אותיות אפשריות

כמויות 3 ומעלה

REGULAR EXPRESSION

⋮ / ^[a-zA-Z]{3,} \$

TEST STRING

Itamar

# עד ולייצירות:

```
from rest_framework import serializers

# serializers convert objects to json and vice versa
from .models import Student
from django.core.validators import MinLengthValidator, RegexValidator

class StudentSerializer(serializers.Serializer):
    name = serializers.CharField(
        max_length=100,
        validators=[
            MinLengthValidator(2),
            RegexValidator(
                regex='^[a-zA-Z0-9]+$',  

                message='name must be alphanumeric',
                code='invalid_name'
            )
        ],
        error_messages = {
            'required': 'Please provide a name for the student',
        }
    )
    id = serializers.IntegerField(read_only=True)

    def update(self, instance, validated_data):
        instance.name = validated_data.get('name', instance.name)
        instance.save()
        return instance

    def create(self, validated_data):
        return Student.objects.create(**validated_data)
```

# בקשות id PUT/DELETE/GET By id

```
from django.urls import path
from .views import index, edit_student

urlpatterns = [
    path('students/', index, name='students'),
    path('students/<int:id>', edit_student, name='students'),
]
```

```
@api_view(['GET', 'PUT', 'DELETE']) #/students/3
def edit_student(request: Request, id:int):
    if request.method == 'GET':
        student = Student.objects.get(id=id)
        serializer = StudentSerializer(student)
        return Response(serializer.data)

    if request.method == 'DELETE':
        student = Student.objects.get(id=id)
        student.delete()
        return Response({'message' : 'Student Deleted'}, status=204)

    if request.method == 'PUT':
        student = Student.objects.get(id=id)
        serializer = StudentSerializer(student, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=400)
```

# טיפ נחמד מהדוקומנטציה: עובדת עם הפרויקט דרך הטרמינל:

```
python manage.py shell
```

# מודל חדש לדוגמאות הבאות:

lec3/api/models.py

```
from django.db import models

class Kitten(models.Model):
    name = models.CharField(max_length=100)
    breed = models.CharField(max_length=100)
```

כדי לעדכן את מסד הנתונים - נבנה מיגרציה וניישם אותה:

**python manage.py makemigrations api**

**python manage.py migrate**

# ModelSerializer:

- במקום שnicדור Serializer במלואו כפי שלמדנו -  
DRF יכולה לייצר אותו עבורנו (:

הספריה תמשח עבורנו Serializer לפי ההגדרות במודל:

lec3/api/serializers.py

```
from rest_framework import serializers
from api.models import Kitten

class KittenSerializer(serializers.ModelSerializer):

    class Meta:
        model = Kitten
        # fields = '__all__'
        fields = ['id', 'name', 'breed']
        # exclude = ['id']
```

# עוד קומפוננטה של DjangoRestFramework Class Based Views:

lec3/api/views.py

```
from rest_framework.views import APIView

class KittenViews(APIView):
    def get(self, request):
        # handle get request
        return Response({"message": "kittens"})

    def post(self, request):
        # handle the request
        return Response({"message": "kittens"})
```

ארגון של הקוד באופן מונחה עצמים:

מחלקה עם متודות  
במקום פונקציה אחת משפטית תנאי:

מיימוש של מחלקה שיורשת מview  
הmethodes שניתן לדרוס:

()get  
()put  
()delete  
()post

lec3/api/urls.py

```
from django.urls import path
from .views import index, edit_student, KittenViews

urlpatterns = [
    path('students/', index, name='students'),
    path('students/<int:id>', edit_student, name='student actions'),
    path('kittens/', KittenViews.as_view(), name='kittens')
]
```

# APIView:

סדר מונחה עצמים של הקוד שכתבנו קודם בקוד Function Based Views

```
class KittenViews(APIView):

    def get(self, request):
        kittens = Kitten.objects.all()
        serializer = KittenSerializer(kittens, many = True)
        return Response({"kittens": serializer.data})

    def post(self, request):
        kitten = KittenSerializer(data = request.data)

        if kitten.is_valid():
            kitten.save() # will invoke create of the serializer
            return Response(kitten.data, status = 201)
        else:
            return Response({"errors": kitten.errors}, status = 400)
```

בשיעורים הבאים נראה  
שים מחוקות שירושות מview ומוסיפות מימוש אוטומטי לMETHODS

# השוואה בין APIView לבין Function Based View

```
@api_view(['GET', 'POST'])
def index(request:Request):

    if request.method == 'POST':
        serializer = StudentSerializer(data = request.data)
        if serializer.is_valid():
            serializer.save() # will invoke create
            return Response(serializer.data, status=201)
        return Response(serializer.errors, status=400)

    students = Student.objects.all()
    serializer = StudentSerializer(students, many=True)
    return Response({'students': serializer.data})
```

מה יותר מסודר בעיניכם :

OOP VS Functional Programming :-)

```
class KittenViews(APIView):

    def get(self, request):
        kittens = Kitten.objects.all()
        serializer = KittenSerializer(kittens, many = True)
        return Response({"kittens": serializer.data})

    def post(self, request):
        kitten = KittenSerializer(data = request.data)

        if kitten.is_valid():
            kitten.save()
            return Response(kitten.data, status = 201)
        else:
            return Response(kitten.errors, status = 400)
```

# שיעור בית:

צרו API שמחזיר מידע על :Fun Facts

מודל:

FunFact:

תכונות:

text, source, source\_url, language, permalink

יש לממש את כל פעולות CRUD

כפי שמיישנו בכיתה

דוגמא לAPI כזה:

<https://uselessfacts.jsph.pl/api/v2/facts/random>



The screenshot shows a Chrome browser window with the following details:

- Address Bar:** uselessfacts.jsph.pl/api/v2/facts/random
- Content Area:** Displays the JSON response from the API call. The JSON object has the following structure:

```
1 // 20250218162407
2 // https://uselessfacts.jsph.pl/api/v2/facts/random
3
4 {
5     "id": "57cb85dcf3078e5b67b63ddc3e733929",
6     "text": "The word \"set\" has more definitions than any other word in the English language.",
7     "source": "djtech.net",
8     "source_url": "http://www.djtech.net/humor/useless_facts.htm",
9     "language": "en",
10    "permalink": "https://uselessfacts.jsph.pl/api/v2/facts/57cb85dcf3078e5b67b63ddc3e733929"
11 }
```