Django

TomerBu

נושאים להיום:

Django Rest Framework
מבוא
התקנה והגדרות
Request Response Classes
Serializers
Function Based Views

שיעורי בית:

צרו את המודלים הבאים:

1

```
Category:
    name
```

שם הקטגוריה - טקסט קצר

קטגוריות לדוגמא:
תכנות, פייתון, לולאות, HTTP, וכו'...

2

```
Post:
  title
  content
  created
  modified
  categories
```

קשר של רבים לרבים
ManyToManyField

3

```
Author:
  birthday
  user
```

קשר של אחד לאחד

4

```
Comment:
  author
  text
  created
  post
```

קשר של אחד לרבים:
models.ForeignKey

קשר של אחד לרבים:
models.ForeignKey

זרעו את המודלים במסד הנתונים

שימו לב ליצור ולידציות

שיעורי בית:

```
django-admin startproject lec3
```

נפתח את התיקיה lec3 בvscode

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

חלונות

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

מאק

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

התקנות

```
pip install django
```

יצירת app

```
python manage.py startapp hw
```

⟶

```
INSTALLED_APPS += ['hw']
```

Hacker.U by ThriveDX

שיעורי בית:

```python
from django.db import models

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

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

```python
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}'
```

```python
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
```

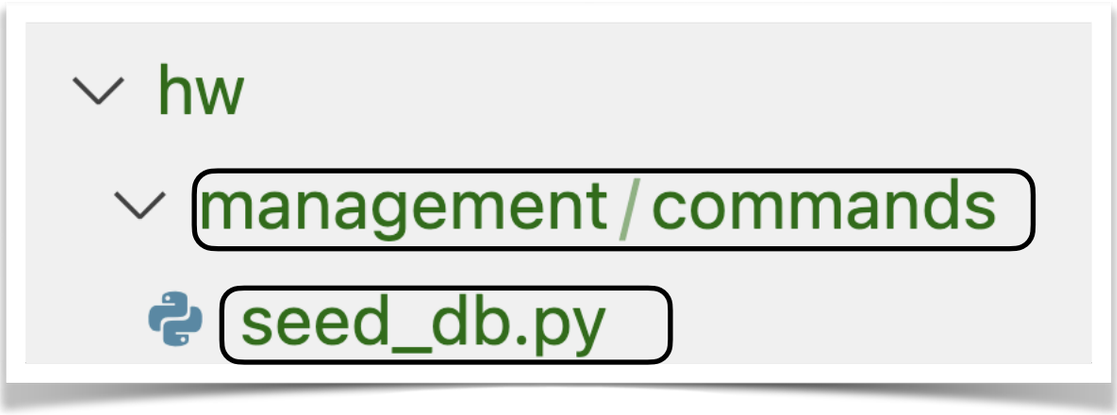
שיעורי בית:

```python
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
```

```python
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
```

שיעורי בית:



hw/management/commands/seed_db

```python
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
```

שיעורי בית:

```python
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')
        user.save()
```

הדפסה בצבע

הצפנת הסיסמא

שמירה נוספת

Hacker.U
by ThriveDX

שיעורי בית:

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

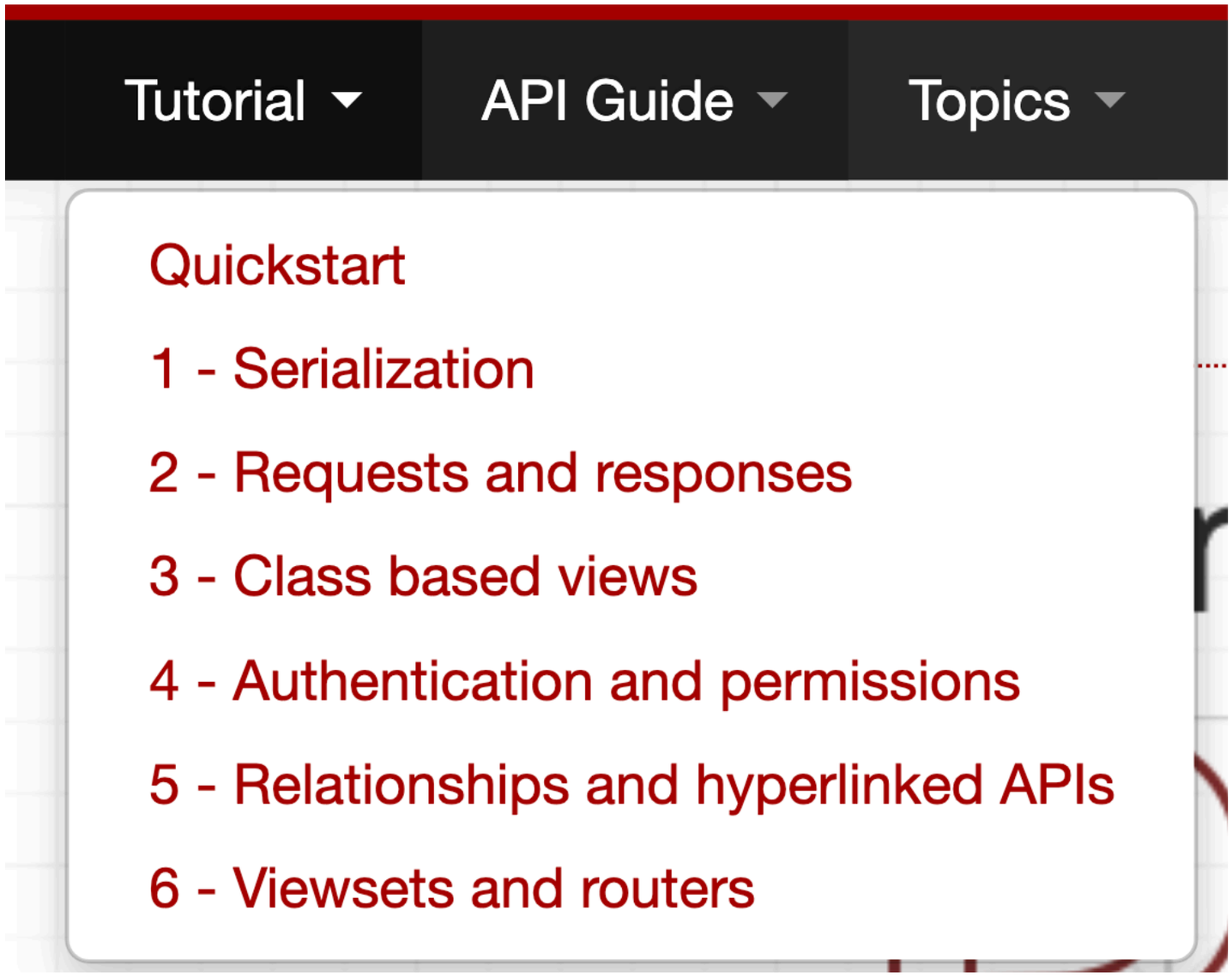
```python
from django.db import migrations
from django.core.management import call_command

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
```

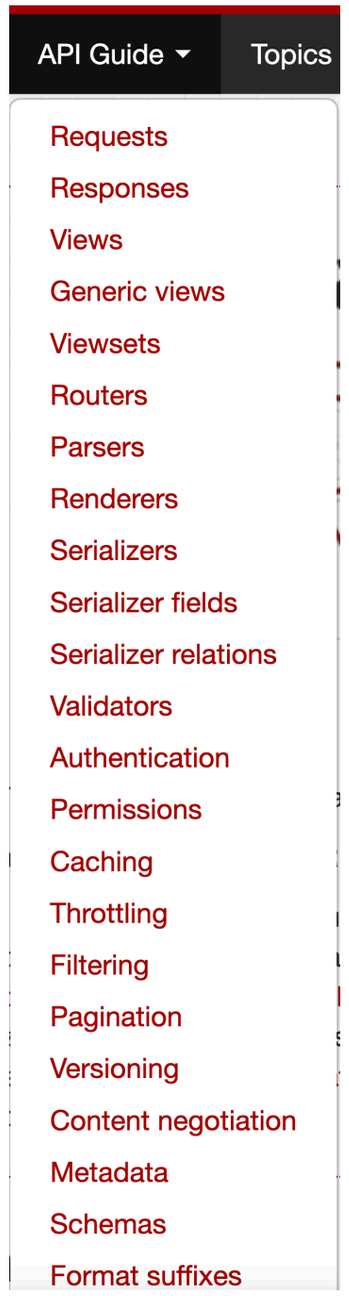
python manage.py migrate hw

python manage.py runserver

django-rest-framework

https://www.django-rest-framework.org/





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

בדיקה:

http:localhost:8000/api/students

lec3/api/urls.py

ניצור קובץ urls

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

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

נערוך את קובץ urls של lec3

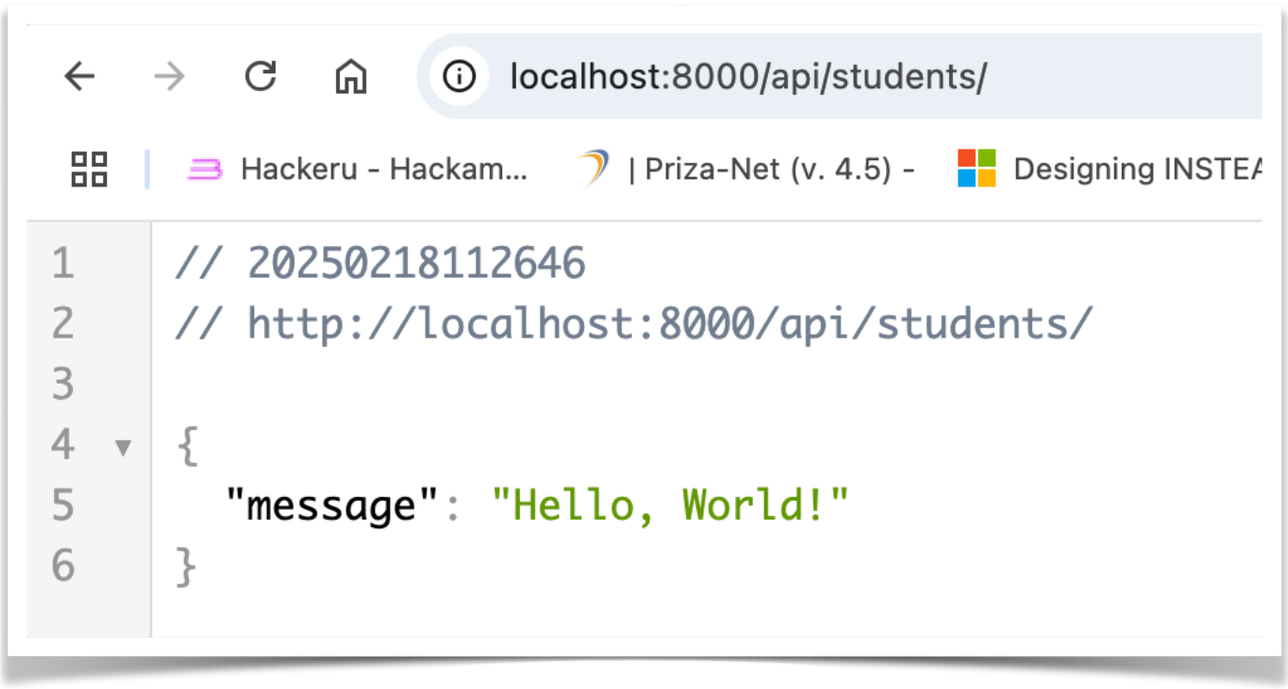
lec3/urls.py

```
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

```python
from django.shortcuts import render
from django.http import HttpResponse, JsonResponse

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



django-rest-framework
מבוא לצורך:

```python
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 defau
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 השנות:
בבקשת HTTP נשלח JSON או XML

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

סריליזציה:
תרגום של אובייקט משפת תכנות לפורמט רצוי (דוגמא JSON/XML)

דה-סריליזציה:
פענוח של XML/JSON 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

נגדיר Serializer שיודע לתרגם אובייקט Student לJSON:

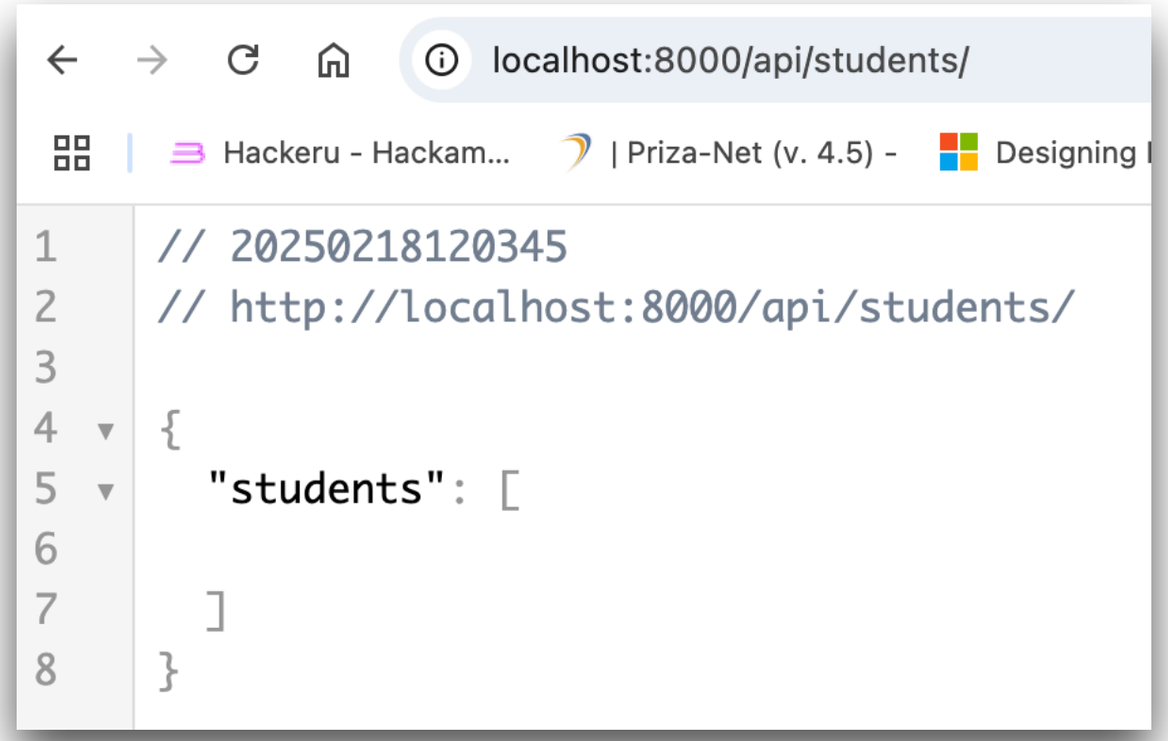
```python
from rest_framework import serializers

# serializers convert objects to json and vice versa

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

```python
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 הStudentSerializer

JSON

Serialization and De-Serialization:



Serialization and De-Serialization:

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

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

```python
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 מבקשה של POST
לטובת יצירת מופע חדש - לשם כך קיימת המתודה create()

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

```python
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

DRF סורקת את ה View שלנו
ומייצרת עבורנו דוקומנטציה:

כדי שזה יעבוד:
יש לוודא שמחזירים Response
ולא הטיפוסים של Django

```python
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})
```

Hacker.U
by ThriveDX

Browsable API:

התמודדות עם בקשת POST

דה-סריליזציה:
מקבל JSON מהלקוח וממיר לאובייקט

```python
@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})
```

סטטוס 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 t
Index resource

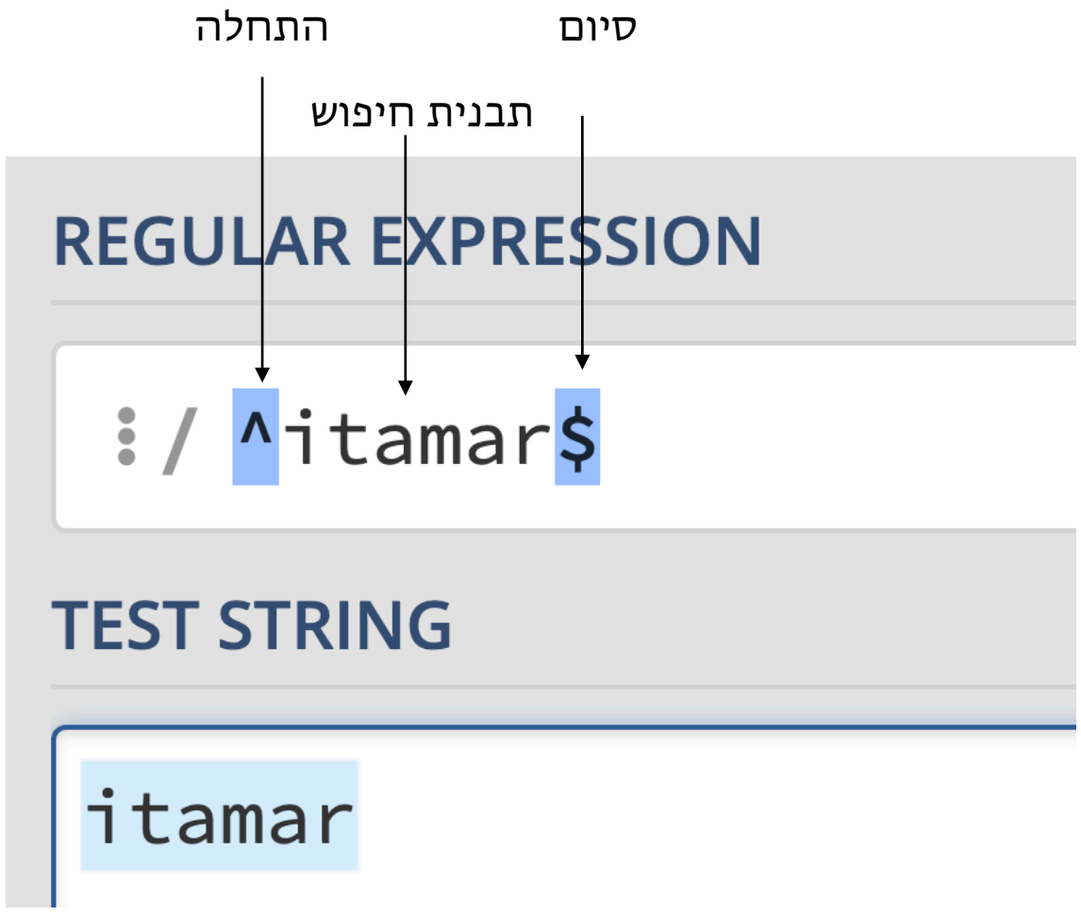
POST

Hacker.U
by ThriveDX

בדיקת תקינות של נתונים בבקשה:

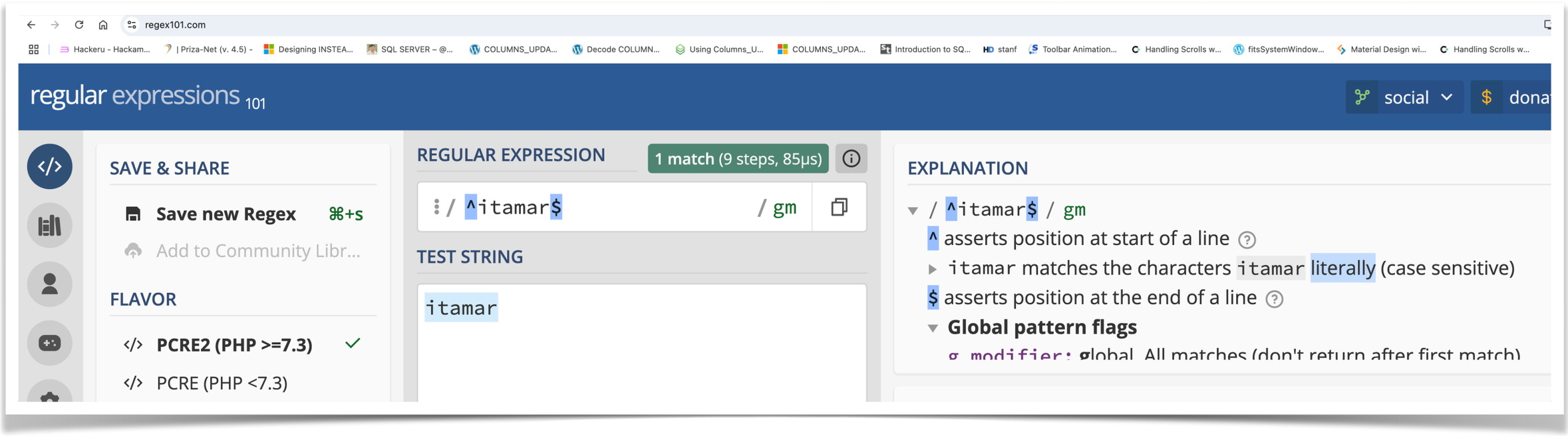
```python
from rest_framework import serializers

# 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:

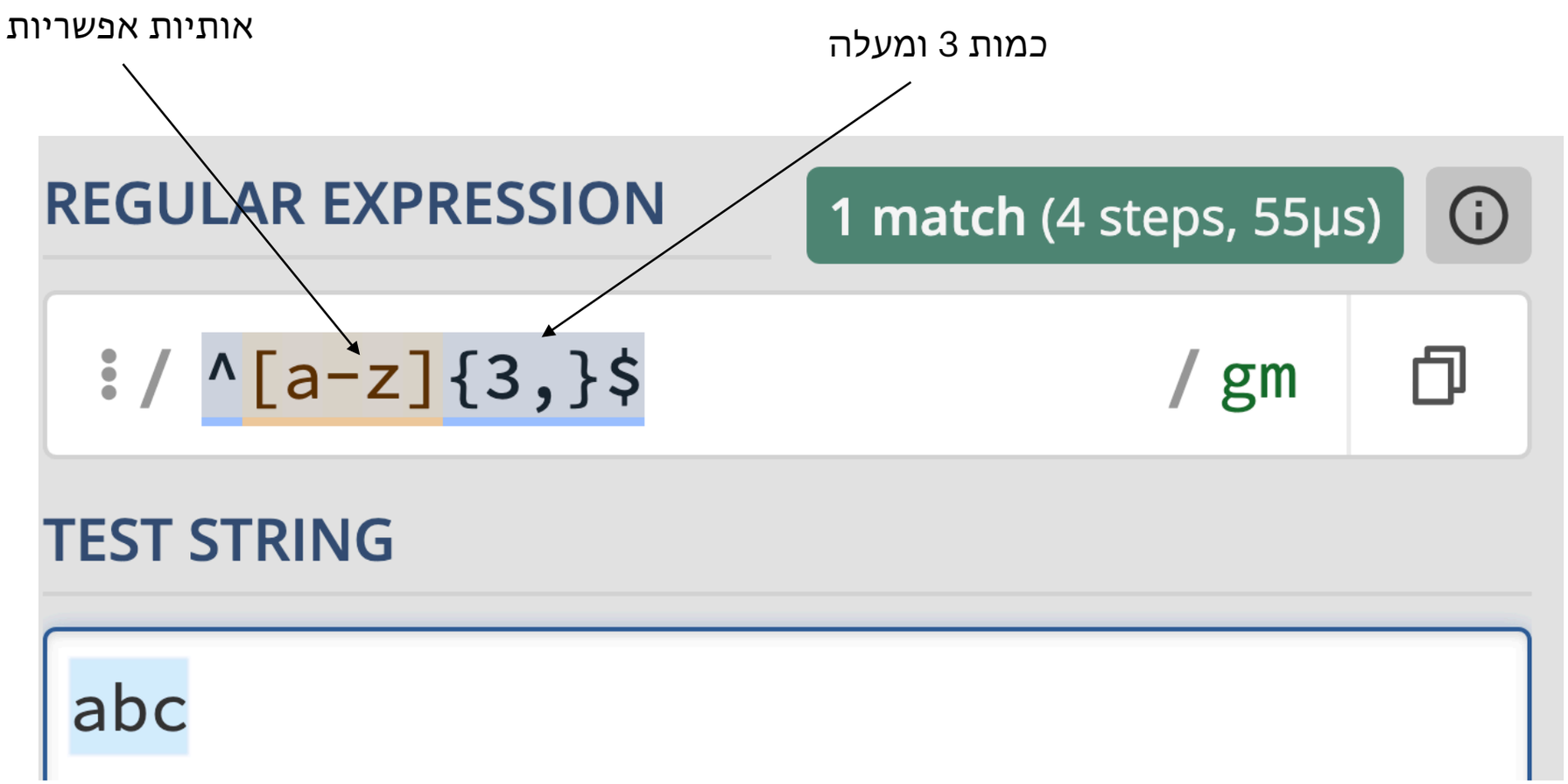


Hacker.U
by ThriveDX

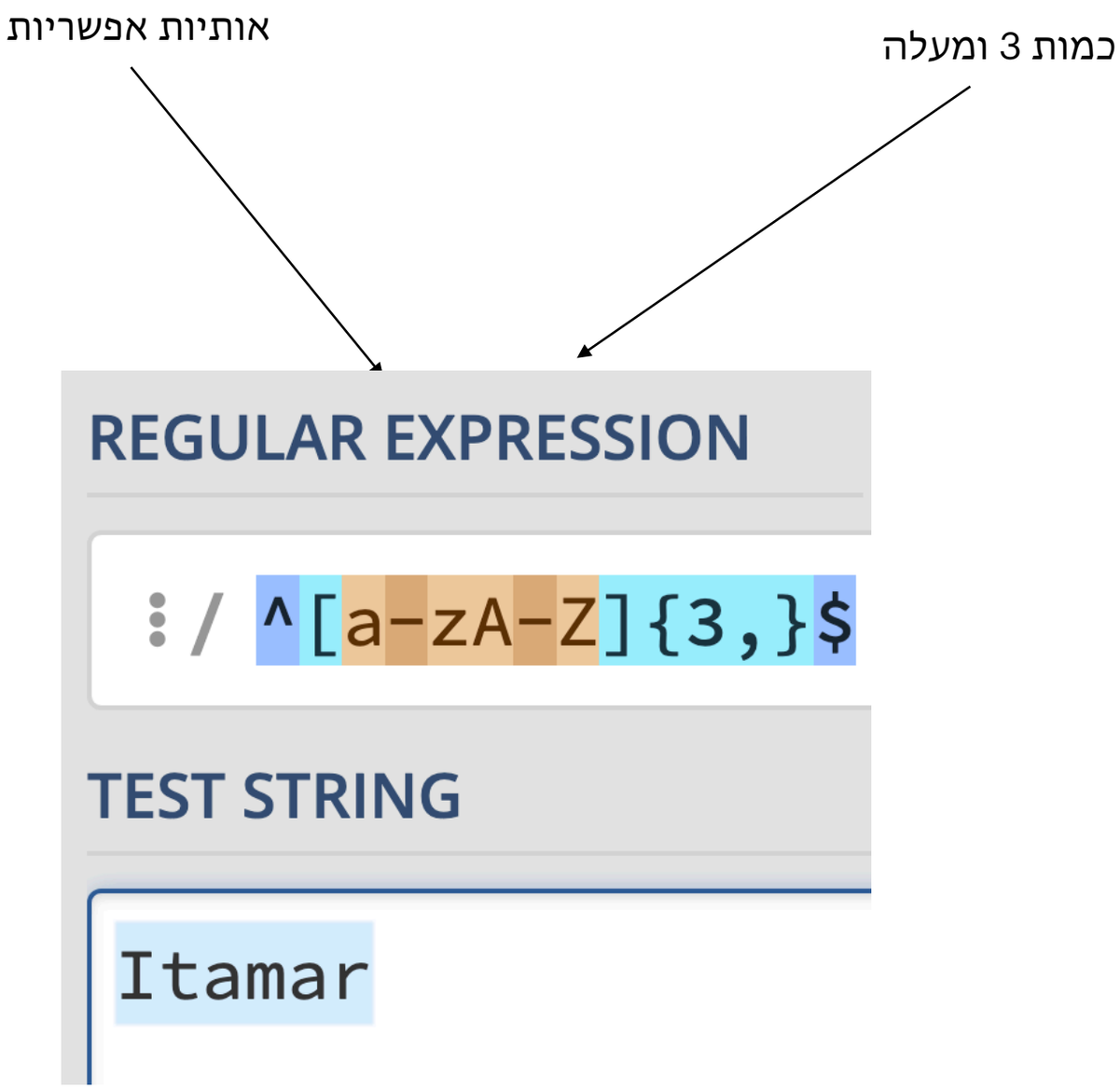
מבוא לביטויים רגולריים:



מבוא לביטויים רגולריים:



מבוא לביטויים רגולריים:



עוד ולידציות:

```python
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)
```

PUT/DELETE/GET By id בקשות

```python
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'),
]
```

```python
@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)
```

Hacker.U
by ThriveDX

טיפ נחמד מהדוקומנטציה:

עבודה עם הפרוייקט דרך הטרמינל:

```
python manage.py shell
```

מודל חדש לדוגמאות הבאות:

lec3/api/models.py

```python
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 שניצור Serializer במלואו כפי שלמדנו -
DRF יכולה ליצור אותו עבורנו (:

הספריה תממש עבורנו Serializer לפי ההגדרות במודל:

lec3/api/serializers.py

```python
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

```python
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 מתודות
במקום פונקציה אחת משפטי תנאי:

מימוש של מחלקה שיורשת מAPIView
המתודות שניתן לדרוס:

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

lec3/api/urls.py

```python
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:

סידור מונחה עצמים של הקוד שכתבנו קודם בFunction Based Views:

```python
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
שיש מחלקות שיורשות מAPIView ומוסיפות מימוש אוטומטי למתודות

השוואה בין APIView
לבין Function Based View:

```python
@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})
```

```python
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)
```

מה יותר מסודר בעיניכם :)

OOP VS Functional Programming :-)

שיעורי בית:

צרו API שמחזיר מידע על Fun Facts:

מודל:
FunFact:
תכונות:
text, source, source_url, language, permalink

יש לממש את כל פעולות הCRUD
כפי שמימשנו בכיתה

דוגמא לAPI כזה:
https://uselessfacts.jsph.pl/api/v2/facts/random