הרצאה 4:

עוד קצת על הפקודה docker exec

הפקודה הזו בעצם מבצעת הרצה של פקודות על קונטיינר רץ. הפקודות ירוצו על התיקיה הראשית של הקונטיינר.
למשל:

```
$ docker exec -d ubuntu_bash touch /tmp/execWorks
```

הפקודה הזו תגרום לקונטיינר ליצור בתוך תיקיית tmp את הקובץ execWorks.

בחזרה למיקרוסרוויסים

כפי שכבר ציינו, מיקרוסרוויסים מתקשרים ביניהם על ידי חשיפת API שבאמצעותו ניתן לגשת למיקרוסרוויס ולשלוח בקשות HTTP לAPI של מיקרוסרוויס אחר.

כל מיקרוסרוויס מתפקד גם כserver המאזין לבקשות וגם כclient כאשר זקוק למשאבים משרתים אחרים.

תזכורת: **HTTP methods**

- GET
- POST
- PUT
- HEAD
- DELETE
- PATCH
- OPTIONS
- CONNECT
- TRACE

כל תגובה מהשרת מגיעה עם status code שמצביע על הצלחה/כשלון של עיבוד הבקשה של הלקוח.

HTTP Error Codes

- Informational responses (100 – 199)
- Successful responses (200 – 299)
- Redirection messages (300 – 399)
- Client error responses (400 – 499)
- Server error responses (599 – 500)

דרכים שונות לשלוח בקשות HTTP

באמצעות דפדפן: בתוך הAddress Bar של הדפדפן אנו בעצם שולחים בקשות GET. לא ניתן לשלוח בקשות POST בדרך זו כי נדרש body שלא ניתן לספק באמצעי זה.

באמצעות פקודת curl בלינוקס:

```
curl https://www.geeksforgeeks.org
```

באופן דיפולטיבי, פקודת curl שולחת בקשות GET.

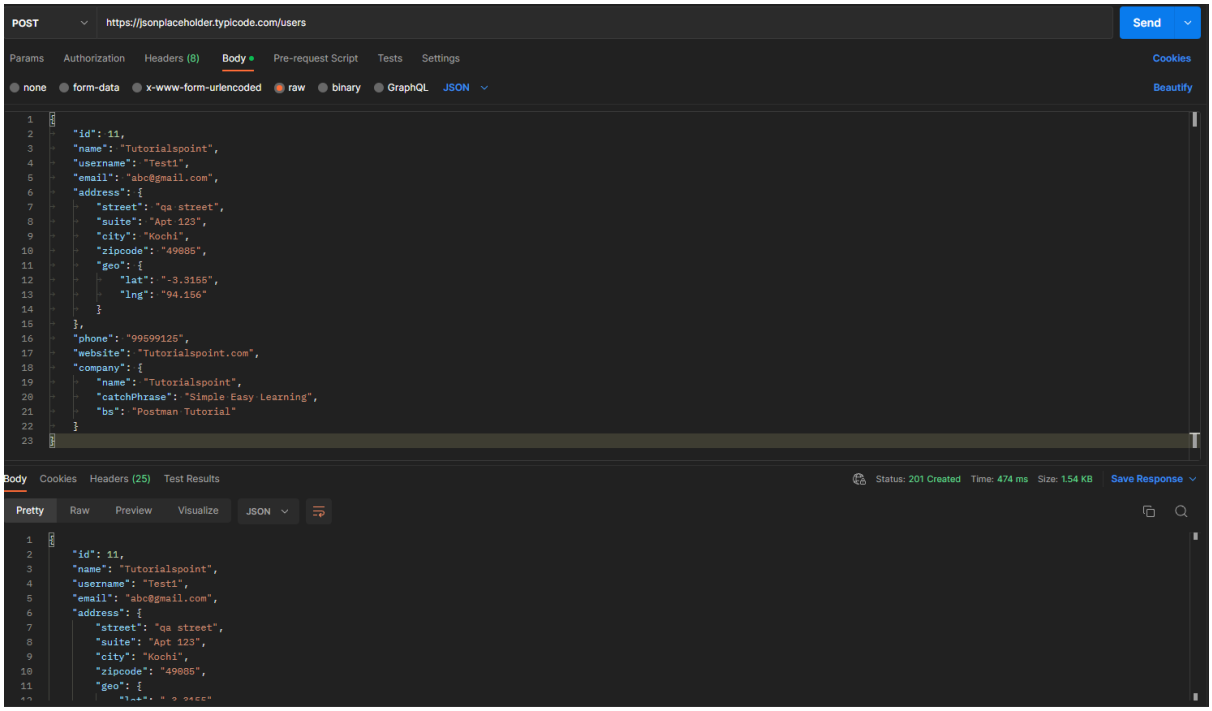
ניתן לשלוח בקשת POST באופן הבא:

Curl POST Request Example

```
curl -X POST https://reqbin.com/echo/post/json
   -H "Content-Type: application/json"
   -d '{"Id": 79, "status": 3}'
```

באמצעות Postman: Postman זוהי מערכת נוחה שבאמצעותה ניתן לשלוח בקשות HTTP.

ניתן להוריד אותה למחשב או להשתמש בגרסה הוובית שלה.



באמצעות פייתון: באמצעות ספריית httpx ניתן לשלוח בקשות HTTP:

```
>>> r = httpx.post('https://httpbin.org/post', data={'key': 'value'})
```

כמו כן אפשר גם באמצעות ספריית requests:

```
>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
```

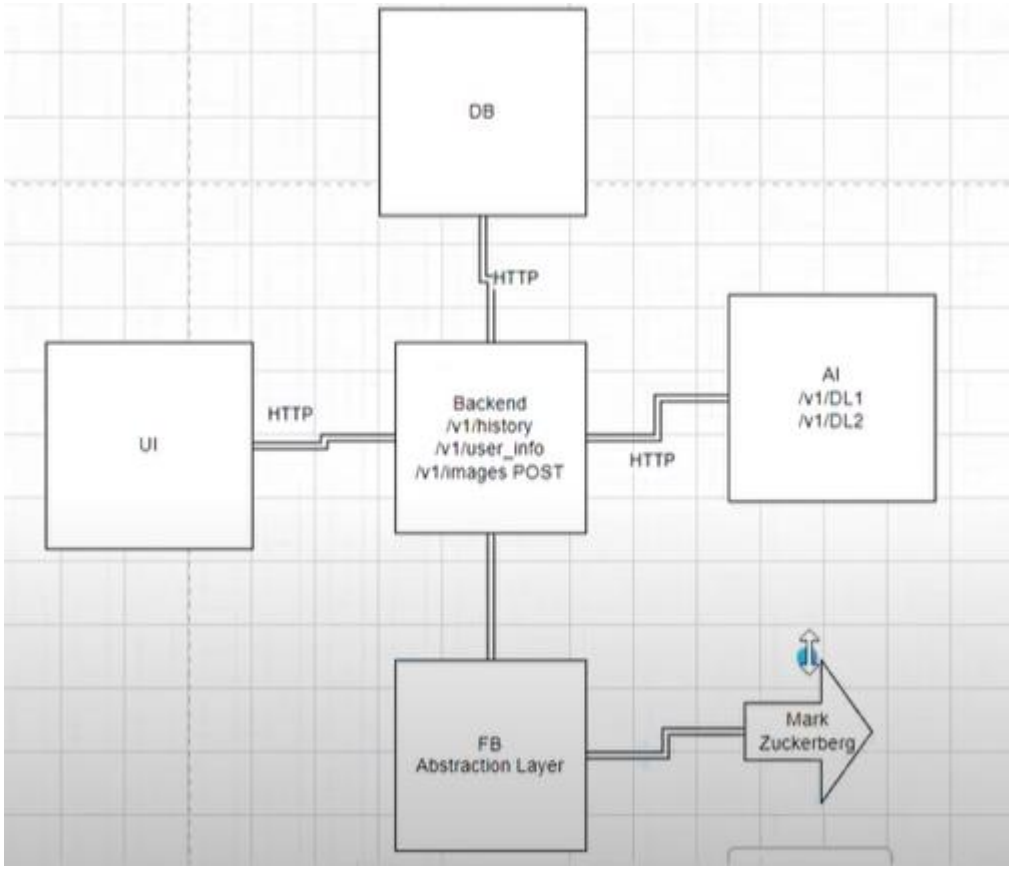
באמצעות קונסול: בדפדפן ישנה האופציה להיכנס לDevelopers tools ולהריץ פקודת fetch של JavaScript דרך הקונסול:

```
fetch('http://example.com/movies.json')
  .then((response) => response.json())
  .then((data) => console.log(data));
```

הדרך שבה אנו נתרגל בניית מערכת בגישה המיקרוסרוויסית

- המערכת שלנו תחולק לשלושה מיקרוסרוויסים לפחות שכל אחד מהם יתפקד בstandalone service.
  רעיונות נפוצים לחילוק המערכת: Backend + Frontend + Database/Cache/AI/ML
- לכל מיקרוסרוויס יהיה docker image משלו. בדרך כלל לכל מיקרוסרוויס יש Dockerfile שבתוכו הimage הבסיסי והתוספות שלנו
- לצורך הרמת מערכת מתפקדת במלואה עם כל חלקיה, ניצור docker-compose אשר יריץ את כל הקונטיינרים במעטפת אחת

המחשה של מערכת שעשינו בביתה לפייסבוק:



Localhost

ברשתות מחשב, localhost זהו הhostname שמתייחס למחשב הנוכחי אשר פונה לעצמו.
בדרך כלל, נשתמש בlocalhost וportsי שזמינים אצלנו במחשב לצורך הרמת מערכת ועבודה על הפיתוח שלה ללא צורך בגישה לשרת חיצוני.

FastAPI

זהו framework לצורך פיתוח RESTful APIs בפייתון.

דוגמה לקוד פשוט העושה שימוש בFastAPI:

code of the server (main.py)

```python
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
async def get_root():
    return {"message": "Hello World", "method": "GET"}

@app.post("/")
async def post_root():
    return {"message": "Hello World", "method": "POST"}
```

כאן בעצם אנו מגדירים שתי בקשות API:

- בקשת GET שמחזירה {"message": "Hello World", "method": "GET"}
- בקשת POST שמחזירה {"message": "Hello World", "method": "GET"}

הendpoint של שתי הבקשות הוא "/". כלומר אם נעלה את השרת לוקלי בפורט 8080 אם נריץ בדפדפן: localhost:8080/ נקבל את התשובה שרשמנו למעלה.

Uvicorn

זהו שרת ASGI (Asynchronous Server Gateway Interface) המאפשר להריץ שרת העובד עם FastAPI

דוגמה לפקודה שנריץ על מנת להרים את השרת הלוקלי על פורט 8080:

```
uvicorn main:app --reload
```

כאן בעצם אנחנו מריצים את הקלאס main.py בתור אפליקציה על פורט 8080.