

פרק 3

תקשורת עם המכונה - REST APIs ויסודות NOS

מטרות הלמידה

בפרק זה נלמד:

- הבנת עקרונות REST APIs לתקשורת עם שירותים בינה מלאכותית
- שיליטה ב-JSON כפורמט חילופי נתונים מרכזי
- יכולת לקרוא ולהבין תיעוד API טכני
- הכרת קודי תגובה והשלכותיהם העסקיים
- ניהול מגבלות קצב (Rate Limiting) ואבטחת מפתחות

3.1 פרולוג: שיחת ולא הובנה

דמיינו מנהל פרויקטים בחברת סטארטאפ, יושב בפגישה עם הצוות הטכני. המפתחת הראשית מציגה תוכנית לשילוב מודול שפה במערכת תמיית הלקוחות. היא מדברת על "POST requests", על "JSON payloads", על "401 errors" ו- "rate limits". המנהל מהנהן בראשו, אך במוחו מתנהל דיאלוג פנימי שונה לאחרי: "אני מבין שהוא חשוב, אבל מה זה בעצם אומר? איך זה משנה על העליות? על הזמן? על הסיכון?"

זהו התסכול של מנהלים רבים בעידן הבינה המלאכותית. הטכנולוגיה מבטיחה, אך השפה שבה היא מותאמת יותר זרה. הפרק הזה בא לגשר על הפער זהה. לא נהפוך אתכם למתחנים, אבל נעניק לכם את הכלים להבין איך המכונות מדברות זו עם זו - ומדוע זה חשוב לכל החלטה עסקית שתקבלו.

3.2 השפה שהמכונות מדברות: מבוא ל-REST API

כשאנו מדברים על שילוב בינה מלאכותית במערכות עסקיות, אנו מדברים למעשה על שיחת לא שיחת בשפה האנושית הרגילה, אלא פרוטוקול תקשורת מדויק ומובנה. זהו REST API - Representational State Transfer Application Programming Interface.

3.2.1 מהו IPA? אנלוגיה למלצר במסעדת

דמיינו מסעדה יוקרטית. אתם, הלקוח, יושבים בשולחן. במטבח עומד שף מוכשר - זהו שירות הבינה המלאכותית, נניח OpenAI. אבל אתם לא יכולים פשוט להיכנס למטבח ולבקש מהשף שירות. בין הלקוח לשף עומד המלצר - זהו ה-API. המלצר מבצע מספר תפקדים קריטיים:

1. **מקבל את הזמןתובע** - הבקשה שלכם (Request)

2. **בודק שהזמן תקין** - האם אתם מורשים לשבת כאן? האם יש לכם כסף לשלם?

3. **מעביר למטבח** - שולח את הבקשה לשירות

4. **מביא את המנה** - מחזיר את התגובה (Response)

5. **מודיע אם שהוא השتبש** - "המטבח עמוס", "המנה אזהה", "כרטיס האשראי נדחה"

ב-REST API, כל אינטראקציה עוקבת אחר המבנה זהה: אתם שולחים בקשה מובנית, והשירות מוחזר תגובה מובנית.

3.2.2 ארבעת הפעולות הבסיסיים: ETELED, TUP, TEG, TSOP

REST מbasס על ארבעה פעלים (HTTP Methods) המייצגים פעולהות שונות:

Method	משמעות	דוגמה עסקית
GET	קריאה מידע	שליפת היסטוריית שיחות עם לקוחות מה-CRM
POST	יצירת משהו חדש	שליחת שאלת חדשה למודל השפה וקבלת תשובה
PUT	עדכון משהו קיים	עדכון הגדרות של סוכן AI קיים
DELETE	מחיקת משהו	מחיקת היסטוריית שיחה רגילה

טבלה 3.1: ארבעת פעלי ה-HTTP ושימושיהם

דוגמה מעשית: נניח שאתם בונים צ'אטבוט לתמיכה לקווות. כל פעם שלקוח שולח הודעה:

1. המערכת שולחים תבצע GET לשולף את הקשר השיחה הקודמת

2. תבצע POST לשולח את השאלה החדש ל-OpenAI

3. OpenAI יחזיר תשובה

4. המערכת תבצע POST נוסף לשומר את התשובה במאגר

כל פעולה היא בקשה נפרדת, עם קוד תגובה, זמן ביצוע ועלות.

3.2.3 מבנה הבקשה:

בקשת API היא כמו מעטפה בדואר:

URL (Endpoint) □

<https://api.openai.com/v1/chat/completions>

מטא-מידע על הבקשה: Headers □

Content-Type: application/json □

Authorization: Bearer sk-... □

User-Agent: MyCompany/1.0 □

הנתונים עצם, בפורמט JSON: Body □

```
{  
    "model": "gpt-4",  
    "messages": [  
        {  
            "role": "user",  
            "content": "?מכלש תוריישל תישדוחה תולע יהמ"  
        }  
    ],  
    "temperature": 0.7  
}
```

השלבה **עסקית קריטית**: כל Header יכול להשפיע על התנהוגות השירות. למשל, אם תשכחו לשלוח Authorization, תקבלו שגיאת 401 Unauthorized. אם תגדירו Content-Type שגוי, השירות לא יבין את הנתונים ויחזר 400 Bad Request.

3.3 NOSJ - שפת חילופי הנתונים של האינטרנט

3.3.1 מהו NOSJ ומדוע הוא כל כך נפוץ?

JSON - JavaScript Object Notation - הוא פורמט טקסטואלי לייצוג נתונים מובנים. הוא נפוץ כל כך משום שהוא:

█ קרייא לבני אדם - אפשר להבין אותו גם בלי להיות מתכנת

█ קל לעיבוד מכונות - כמעט כל שפת תכנות יודעת לקרוא ולכתוב אותו

█ גמיש - מאפשר מבני נתונים מורכבים

█ קומפקטי - לא מבזבז תווים מיוחדים (בניגוד ל-XML)

3.3.2 סוגי נתונים ב-NOSJ

JSON תומך בשישה סוגי נתונים בסיסיים:

סוג נתונים	דוגמה	שימוש עסקי
String (מחרוזת)	"Hello World"	שמות, טקסטים, מזהים
Number (מספר)	42, 3.14	עליות, כמות, ציונים
Boolean (בוליאני)	true, false	דגלים, הרשות
Null (ריק)	null	ערך חסר, לא זמין
Array (מערך)	[1, 2, 3]	רשימות, אוסףים
Object (אובייקט)	{"key": "value"}	ישיות מובנות

טבלה 3.2: סוגי הנתונים ב-JSON

3.3.3 מבנים מקוונים: הכוח האמתי של NOS

היכולתukan אובייקטים ומערכות היא מה שהופך את JSON לעוצמתי. הנה דוגמה למבנה ריאלי:

```
{
  "customer": {
    "id": "C12345",
    "name": "מַכְנִיק תּוֹיְגָוָלְוָנָכְט תְּרֵבָח",
    "subscription": {
      "plan": "Enterprise",
      "price_per_month": 5000,
      "currency": "ILS"
    },
    "usage_history": [
      {
        "month": "2025-01",
        "api_calls": 150000,
        "cost": 4823.50
      },
      {
        "month": "2025-02",
        "api_calls": 180000,
        "cost": 5789.20
      }
    ],
    "is_active": true,
    "last_payment": "2025-02-15"
  }
}
```

שימוש לב למבנה המוקון:

customer ☐ הוא האובייקט הראשי

customer ☐ subscription הוא אובייקט מוקון בתוך customer

usage_history ☐ הוא מערך של אובייקטים, כל אחד מייצג חודש

מבט מנהלי: כשאתם מבקרים במצב הפתוח "לשלוף את נתוני השימוש של הלוקה", המערכת ת策רך לנווט דרך המבנה זהה. אם תבקשו "עלות פברואר", היא ת策רך:

1. לגשת ל-**customer**

2. לחפש במרחב **usage_history**

3. למצוא את הרשומה עם "month": "2025-02"

4. לחלץ את **cost**

כל שלב יכול להיכשל (אולי החודש לא קיים?), ולכן קוד איקוני צריך לטפל בכל התרחישים.

3.3.4 תרגול: קריאת JNOS מתיעוד IPA

הנה קטע אמיתי מתיעוד API OpenAI לבקשת :chat completion

```
{  
    "id": "chatcmpl-abc123",  
    "object": "chat.completion",  
    "created": 1706543210,  
    "model": "gpt-4",  
    "choices": [  
        {  
            "index": 0,  
            "message": {  
                "role": "assistant",  
                "content": "... ליעופב שומישב היולדת תישדועה תולעה..."  
            },  
            "finish_reason": "stop"  
        }  
    ],  
    "usage": {  
        "prompt_tokens": 25,  
        "completion_tokens": 50,  
        "total_tokens": 75  
    }  
}
```

שאלות הבנה למנהליים:

1. איפה נמצאת התשובה בפועל של המודל? (رمز: `(choices[0].message.content`)

2. כמה טוקנים נרככו בסך הכל? (75)

3. איך תחשבו את העלות אם מחיר ה-*outpu*t הוא \$0.01 לאלף טוקנים ומה-\$0.03 לאלף?

$$\begin{aligned}
 &= (25 \times 0.01/1000) + (50 \times 0.03/1000) \\
 &= 0.00025 + 0.0015 = \$0.00175
 \end{aligned}$$

זו החשיבה שאתה צריכה לאמץ: כל בקשה היא עלות. כל טוקן נספר. כשאתם מתכוונים מערכת עם מיליון שיחות חודשיות, הבנת המבנה זהה קרייטית.

3.4 קודי תגובה: מה המכונה אומרת לנו?

כל בקשת API מסוימת בקוד תגובה (HTTP Status Code) - מספר תלת-ספרתי שמסכם מה קרה. הבנת הקודים הללו היא הבדל בין ניהול שיוודע לשאלות נכונות לבין מי שמנה בפגישות טכניות.

3.4.1 מפת הקודים - ומה הם אומרים עליהם

קוד	משמעות טכנית	משמעות עסקית
200 OK	הבקשה הצלחה	הכל עובד. זמן לשלם.
400 Bad Request	הבקשה שגויה	הקוד שלכם שלח משהו לא תקין. באג בצד שלכם.
401 Unauthorized	לא מורשה	מפתח ה-API שגוי או פג תוקפו. בעיית אבטחה.
429 Too Many Requests	יותר מדי בקשות	עברתם את מגבלת הקצב. צריך לאט או לשדרג תוכניתה.
500 Internal Server Error	שגיאה בשרת	הבעיה בצד השירות, לא שלכם. צור קשר עם התמיכה.
503 Service Unavailable	שירות לא זמין	שירותים עמוסים או בתחזוקה. נסו שוב מאוחר יותר.

טבלה 3.3: קודי תגובה נפוצים והשלכותיהם

3.4.2 סיפור מהשיטה: מקרה 924

חברת SaaS ביגנונית בישראל שילבה ChatGPT במערכת התמיכה שלה. בהשקה ראשונה, הכל עבדמצוין. יום לאחר מכן, בשעה 00:09 בובוקר - קריסה מוחלטת. הצ'אטבוט חזר עם הודעות שגיאה. המנהלת הטכנית התקשרה לישיבת חירום.

הבעיה? קוד 429. התוכנית שלחם ב-AI OpenAI הייתה מוגבלת ל-60 requests per minute. בובוקר, ככל נציגי השירותים נכנעו למערכת בו-זמנית ושלחו בקשות, המערכת חצתה את המגבלה תוך שניות.

הלקח המנהלי:

1. **תבננו לעומס** - לא רק לשימוש ממוצע
2. **הבינו את התוכנית שלכם** - מה המגבילות? האם הן מתאימות לצרכים?
3. **בנו מנגנון נסיגה (Fallback)** - מה קורה כשה-API לא זמין?

gnitimiL etaR 3.5 - מנהלים ומגבילות

3.5.1 מהו gnitimiL etaR ומהו קיים?

Rate Limiting הוא מנגנון הגנה של שירותיים. הוא מגביל כמה בקשות אתם יכולים לשלוח בפרק זמן נתון. למשל:

□ בקשות לדקה 3: OpenAI Free Tier

□ בקשות לדקה (תליי בתוכנית) 60: OpenAI Pay-as-you-go

□ בקשות לדקה ברמה בסיסית 50: Anthropic Claude

למה? כי אחרת:

1. לקוח אחד יכול "לחנוק" את השירותים לכולם

2. עלויות החומרה יתפוצצו

3. קשה לחזות עומסים ולתכנן תשתיות

3.5.2 חישוב throughput מקסימלי

נניח שיש לכם מגבלה של 60 requests per minute, וכל בקשה לוקחת בממוצע 2 שניות (כולל זמן רשת ויבוד). מהו throughput המקסימלי שלכם?

נוסחה:

$$(3.1) \quad \text{Throughput (requests/min)} = \frac{60}{\text{Latency}_{\text{avg}} (\text{sec})}$$

חישוב:

$$\text{Throughput} = \frac{60}{2} = 30 \text{ requests/min}$$

אבל רגע! המגבלה שלכם היא 60 requests/min. אז למה אתם מוגבלים ל-30? התשובה: ה-latency (זמן התגובה) הוא הגורם המגביל האמתי כאן, לא rate-limit. אם הייתם שולחים את כל 60 הבקשות בפרק תחילת הדקה, היו נגמרות תוכן שנייה, ואז הייתם מחכים 59 שניות לדקה הבאה. לא יעיל.

סטרטגיה נבונה: פיזור הבקשות באופן אחיד לאורך הדקה - בקשה אחת כל שנייה. כך תנצלו את המגבלה בצורה אופטימלית.

3.5.3 תכון ארכיטקטורת etR imitign

כשאתם מתכננים מערכת עם API חיצוני, עלייכם לענות על השאלות הבאות:

1. מהו נפח הבקשות הצפוי?

- ממוצע יומי? שעתשיא?
- דוגמה: $10,000 \text{ שיחות ביום} = 6,944 \text{ שיחות ביום עסקי} (16 \text{ שעות}) = 7.2 \text{ שיחות לדקה ממוצע}$

2. האם המגבלה מספקת?

- אם התוכנית מאפשרת RPM 60 ואתם צריכים רק 7.2 ב ממוצע - מצוין
- אבל מה בשעתשיא? אולי RPM 30? עדין בטוחה

3. מה קורה כשחוצים את המגבלה?

- העמדת בקשות בתור והמתנה Queueing
- ניסיון חוזר אחרי עיכוב Retry Logic
- הצגת הודעת "המערכת עומס", נסה שוב Graceful Degradation

4. מה העלות של שדרוג?

- אצל AI, OpenAI, שדרוג לרמה גבוהה יותר יכול להכפיל את ה-Rate Limit
- צריך לשקל: האם כדאי לשדרוג, או לבנות לוגיקה חכמה יותר?

3.6 IPA syEK ואבטחה - מי שומר על השומרים?

3.6.1 מהו IPA syEK?

API Key הוא מפתח זיהוי ייחודי שמאפשר לשירות לדעת מי שלוח את הבקשה. הוא נראה בערך כך:

sk-proj-abc123def456ghi789jkl012mno345pqrs678stu901vwx234

מפתח זה הוא למעשה **הסימה** לחשבונם שלכם. מי שמחזיק בו יכול:

- לשלוח בקשות בשםיכם
- לצרוך את המכסה שלכם
- לגרום לחיבורים בcredentails האשראי שלכם
- בתרחיש הגרווע - לגשת לנוטונים רגיסטים אם המפתח מאפשר זאת

3.6.2 סיפור אימה: דליפת IPA syEK ב-GitHub

- סטודנט לתוכר שני פיתח בוט Telegram שמשתמש ב-OpenAI. הוא דחף את הקוד ל-GitHub כולל המפתח. תוך 20 דקות, בוטים אוטומטיים שסורקים את GitHub מצאו את המפתח והתחלפו להשתמש בו.

עד שהסטודנט שם לב למחרת בبوك, נצרכו **\$1,200** בחשבון שלו. OpenAI לא החיזרו את הכספי - זו אחוריותו של המשתמש לשומר על המפתח. **לקחת:** IPA Keys הם כמו כרטיסי אשראי. לעולם לא מפרסמים אותם.

3.6.3 מדיניות ניהול IPA syeK בארגון

מנהלים, עליהם לודא שהארגון מישם את העקרונות הבאים:

1. ניהול סודות (Secrets Management)

- אל תשמרו מפתחות בקוד HashiCorp Vault ,Azure Key Vault ,AWS Secrets Manager
- השתמשו בכלים כמו שמרו בקבצי .env. (שלא כללים ב-Git)

2. עקרון הרשות המינימלית (Least Privilege)

- צרו מפתחות שונים למטרות שונות
- דוגמה: מפתח לפיתוח (מגבלה נמוכה), מפתח לייצור (מגבלה גבוהה)
- הגבילו הרשות - מפתח לקריאה בלבד vs קריאה+כתיבת

3. רוטציה תקופתית

- החליפו מפתחות כל 90 ימים
- בעת עזיבת עובד - ביטול מיידי של כל המפתחות שהיו ברשותו

4. ניטור וازעקות

- התראה על שימוש חריג (פטאום 1,000 בקשה לדקה)
- התראה על חריגה מותקציב (עלות יומית עברה סף)
- לוגים מפורטים של כל שימוש

5. תוכנית תגובה לאירוע (Incident Response Plan)

- מה עושים אם מפתח דלף?
- שלב 1: ביטול מיידי של המפתח
- שלב 2: בדיקת לוגים - מה נעשה עם המפתח?
- שלב 3: הנפקת מפתח חדש ועדכון כל המערכות
- שלב 4: דיווח אם יש חובה רגולטורית

3.7 עלויות ומדדדים עסקיים

3.7.1 נוסחת עלות לבקשה

כל בקשה ל-API של שירות LLM עולה כ-\$1. הנוסחה הבסיסית:

$$(3.2) \quad \text{Cost}_{\text{request}} = (\text{Tokens}_{\text{input}} \times \text{Price}_{\text{input}}) + (\text{Tokens}_{\text{output}} \times \text{Price}_{\text{output}})$$

אבל זו רק הערות הישרה. הוצאות האמיתית כוללת:

$$(3.3) \quad \text{Total Cost} = \text{API Cost} + \text{Infrastructure} + \text{Development} + \text{Maintenance} + \text{Support}$$

דוגמה מעשית:

נניח חברה עם 1,000 פניות תמיכה ביום. כל פניה:

- Input : 200 טוקנים (הקשר + שאלת הלקוח)
 - Output : 150 טוקנים (תשובה המודול)
 - מחירי GPT-4 : \$0.03 ל-1K input , \$0.06 ל-1K output
- חישוב לפנייה אחת:

$$\begin{aligned} \text{Cost}_{\text{request}} &= \left(\frac{200}{1000} \times 0.03 \right) + \left(\frac{150}{1000} \times 0.06 \right) \\ &= 0.006 + 0.009 = \$0.015 \end{aligned}$$

לחודש (30 ימים):

$$\text{Monthly API Cost} = 0.015 \times 1000 \times 30 = \$450$$

זה נראה סביר, נכון? אבל הוסיפו:

- שכר מפתח (0.5 FTE) : \$3,000 /חודש
- שירות (AWS) : \$200 /חודש
- ניטור וכליים: \$100 /חודש
- **סה"כ: \$3,750 /חודש**

פתרונות העלות האמיתית פי 8 מהעלות הישירה של ה-API.

3.7.2 תכנון התקציב IPA

במנחים, עלייכם לשאול:

1. מהו נפח הבקשות הצפוי? - הערכה ראשונית + צמיחה
2. מהו אורך הטוקנים הממוצע? - תלוי בתחום
3. האם יש עונתיות? - חודש ינואר עומס יותר?
4. מהי תוכנית החירות? - אם התקציב נגמר באמצעות החודש?

דוגמת מדיניות:

- התקציב חודשי: \$1,000
- התראתה ב-70%: הودעה למנהל
- התראתה ב-90%: הפסקת שירותי לא קריטיים
- ב-100%: חסימה מוחלטת (למנוע חריגה)

3.8 דוגמאות מעשיות מועלם העסקים

3.8.1 דוגמה 1: חיבור אפליקציה פנימית ל-OAnep IPA

תרחיש: חברת SaaS רוצה להוסיף תכונת "סיכום חכם" למערכת CRM שלהם. כל פגישה עם לקוחות תתווד, ובשילובת כפתרו המערכת תיצור סיכום.

שלבי האינטגרציה:

1. רכישת מפתח API

- הרשמה ל-OpenAI
- הגדרת כרטיס אשראי
- יצירת מפתח עם רשותות מתאימות

2. פיתוח הלוגיקה

- שליפת טקסט הפגישה מהמסד נתונים
- בניית prompt: "סכם את הפגישה הבאה בתבלייטים..."
- שליחת בקשה POST ל-<https://api.openai.com/v1/chat/completions>
- קבלת התגובה והציגתה למשתמש

3. טיפול בשגיאות

- אם 401 - המפתח לא תקין, הצג הודעה למנהל מערכת
- אם 429 - נסה שוב אחרי 60 שניות
- אם 500 - שמור את הבקשת ונסה אוטומטית אחר כך

4. ניטור

- לוג של כל בקשה - מתי, מי, כמה טוקנים, כמה עלה
- דashboard חדש: סה"כ בקשות, עלויות, זמן תגובה

תוצאה עסקית:

- חיסכון של 10 דקות לנציג מכירות לאחר כל פגישה
- 20 פגישות ביום □ 10 דקות = 200 דקות = 3.3 שעות
- בשכר של \$30/\$שעה: חיסכון של \$100/\$יום = \$3,000/\$חודש
- עלות API: \$200/\$חודש
- IOR : $\frac{($3,000 - $200)}{200} = 1400\%$

3.8.2 דוגמה 2: שליפת נתונים מ-MRC ושלוב עם MLL

תרחיש: צוות מכירות רוצה "עזרה חכם" שיכל לענות על שאלות כמו "מי הלקוחות שלא קנו מائינו 6 חודשים ושוימים מעל \$50K?"
ארכיטקטורה:

1. שלב 1: שאלתה ל-API CRM

- בקשה GET ל-Salesforce/HubSpot API
- משיכת רשימת לקוחות עם שדות: תאריך רכישה אחרונה, ערך כולל
- פילטור בצד הקוד: רק לקוחות שעוניים לקריטריונים

2. שלב 2: העברה ל-LLM

- הכתנת prompt: "הנה רשימת לקוחות. הצע אסטרטגיה לחזרה אליהם"
- שליחה ל-OpenAI
- קבלת המלצות מותאמות אישית

3. שלב 3: הצגה למשתמש

- פורמט JSON מוחזר כטבלה נאה
- כפורי פעולה: "שלח מייל", "קבע פגישה"
- אתגרים שנתקלו בהם:**
- **Salesforce** של Rate Limiting: 1,000 בקשות ליום. פתרון: קאש יומי של נתונים
- גודל ההקשר: רשיימה של 500 לקוחות = 50K טוקנים. פתרון: סינון מקדים לרלוונטיים ביותר
- **עלויות:** חודש ראשון עלה \$.800. פתרון: מיקרו-קאש של שאלות נפוצות

3.8.3 דוגמה 3: דashboard בזמן אמת של נתונים AI

תרחיש: מנהל טכנולוגי רוצה לראות "מה קורה עכשיו" במערכת ה-AI.
哉דים בדASHBOARD:

- **בקשות לדקה** - האם מתקרבים ל-Rate Limit?
- **עלות שעתיות** - כמה הוציאנו עד עכשיו היום?
- **זמן תגובה** - האם יש האטה?
- **קודי שגיאה** - כמה 429, 500, וכו'
- **טוקנים ממוצעים** - האם המשתמשים שולחים שאלות ארוכות מדי?
ישום טכני:
- כל בקשה נשמרת ב-Database עם timestamp
- מריצ' ארגזיות כל 5 דקות Python script
- ממשק Streamlit מציג את הנתונים בזמן אמת

ערך עסק:

- זיהוי בעיות לפני שהלכות מרגישים בהן
- אופטימיזציה של עלויות - "למה ביום שלישי תמיד עולה פי 2??"
- תכנון קיבולת - "נראה שצריך לשדרג את התוכנית לפני סוף החודש"

3.9 תרגילים

3.9.1 תרגילים תיאורתיים

תרגיל 1.3. פענוח תיעוד IPA ותכנון אינטגרציה
קראו את התיעוד הבא מ-Anthropic Claude API

POST <https://api.anthropic.com/v1/messages>

Headers:

```
x-api-key: YOUR_API_KEY  
content-type: application/json
```

Body:

```
{  
    "model": "claude-3-opus-20240229",  
    "max_tokens": 1024,  
    "messages": [  
        {"role": "user", "content": "Hello, Claude"}  
    ]  
}
```

Response (200 OK):

```
{  
    "id": "msg_123",  
    "type": "message",  
    "role": "assistant",  
    "content": [{"  
        "type": "text",  
        "text": "Hello! How can I assist you today?"  
    }],  
    "usage": {  
        "input_tokens": 12,  
        "output_tokens": 20  
    }  
}
```

משימה:

1. זהו את כל רכיבי הבקשה: URL, Method, Headers, Body

2. הסבירו מה תפקיד כל שדה ב-Body

3. חשבו את העלות אם $\text{Output} = \$75/\text{million tokens}$, $\text{Input} = \$15/\text{million tokens}$

4. תכננו: איך תטמיעו זאת באפליקציה צ'אט? אילו שגיאות אפשריות?

תרגיל 2.3. ניתוח קוזי שגיאה וכטיבת נהיל טיפול

הארגון שלכם מפתח צ'אטבוט פנימי. לאחר שבוע בייצור, אלו התקלות שדווחו:

□ 401 Unauthorized - 15 מקרים

□ 429 Too Many Requests (בעיקר בשעה 9-10 בלילה) - 47 מקרים

□ 500 Internal Server Error - 3 מקרים

□ 400 Bad Request - 8 מקרים

משימה:

1. לכל קוד שגיאה, הציעו סיבה אפשרית

2. כתבו נהיל טיפול לכל אחד (מה המערכת צריכה לעשות אוטומטית? מתי להתריע לאדם?)

3. הציעו שניים ארכיטקטוניים למניעת ה-429 בשעות השיא

תרגיל 3.3. תכנון ארכיטקטורת RetaL imitigngn

ארגון עם 500 עובדים רוצה להטמייע עוזר AI פנימי. ההערכות:

□ 30% מהעובדים ישתמשו يوم-יום

□ כל משתמש ישלח בממוצע 10 שאלות ביום

□ יום עבודה: 8 שעות

□ שעת שיא: 60% מהשימוש מתרცז ב-2 שעות (9-11)

התוכנית הנוכחית: .60 requests/min.

משימה:

1. חשבו את ממוצע הביקשות לדקה ביום רגיל

2. חשבו את שיא הביקשות בשעת העומס

3. האם התוכנית מספיקה? אם לא, מה צריך?

4. הציעו 3 אסטרטגיות להתמודדות ללא שדרוג מייד

תרגיל 4.3. כתיבת מדיניות ניהול IPA sys

אתם סמנכ"ל טכנולוגיה של חברת שימושה ב-5 שירותים API שונים: OpenAI, AWS, Twilio, Stripe, SendGrid,

משימה: כתבו מדיניות ארגונית בת 2 עמודים שמכסה:

1. מי מורשה ליצור מפתחות?

2. איך מאחסנים אותן? (אסור לכתוב "בקוד")

3. כמה זמן מפתח תקין?

4. מה קורה כשעובד עוזב?

5. מה התחליך אם מפתח דף?

6. איך מנטרים שימוש?

תרגיל 5.3. חישוב עלויות IPA לתרחישי שימוש
חברה שוקלת 3 תרחישי שימוש שונים ב-GPT-4:
תרחיש A: סיכום דוחות יומי

□ 50 דוחות ביום

□ כל דוח: 3,000 טוקנים input

□ כל סיכום: 300 טוקנים output

תרחיש B: צ'אטבוט תמייה

□ 200 שיחות ביום

□ כל שיחה: 500 טוקנים input, 200 טוקנים output

תרחיש C: ניתוח חוות

□ 10 חוות ביום

□ כל חוות: 8,000 טוקנים input, 1,000 טוקנים output

מחירי GPT-4: Output \$0.06/1K, Input \$0.03/1K
משימה:

1. חשבו עלות חודשית (22 ימי עבודה) לכל תרחיש

2. דרגו מהיקר לאול

3. אם התקציב הוא \$500/חודש, אילו תרחישים אפשריים?

4. הציעו אופטימיזציה לתרחיש היקר ביותר

3.9.2 קוד nohtyP

תרגיל 6.3. שליחת בקשה ל-OpenIA API וקבלת תשובה
כתבו תוכנית Python שמבצעת את הפעולות הבאות:

1. טוענת את מפתח ה-API מקובץ env. (לא מהקוד!)

2. שולחת בקשה ל-OpenAI Chat Completions API

3. מקבלת תשובה ומציגה אותה

4. טיפול בשגיאות: 401, 429, 500

5. הצגת מספר הטוקנים שנוצלו

קוד בסיס:

```
import os
import requests
from dotenv import load_dotenv
```

TODO: דוגמה תא ומיילשה:

דרישות:

השתמשו ב ספריות: `python-dotenv`, `requests`

הקוד צריך להיות ברור וקריא

הוסיפו הערות מסבירות

תרגיל 7.3. ניהול ליטראות עם גנטים
כתבו מחלקה `APIClient` שמטפלת אוטומטית ב-Rate Limiting

1. אם מקבל 429, המתן 60 שניות ונסה שוב

2. מספר ניסיונות מקסימלי: 3

3. תיעוד (logging) של כל ניסיון

4. אם נכשל 3 פעמים, זרוק חריגה

קוד בסיס:

```
import time
import logging
import requests

class APIClient:
    def __init__(self, api_key, base_url, max_retries=3):
        self.api_key = api_key
        self.base_url = base_url
        self.max_retries = max_retries

    def make_request(self, endpoint, data):
        # TODO: הקיגולה מושוו
        pass

    # שומיש תמגוד:
    client = APIClient(api_key="sk-...",
                        base_url="https://api.openai.com/v1")
    response = client.make_request("/chat/completions", {...})
```

בונוס:

הוסיפו exponential backoff (המתנה גדלה עם כל ניסיון)

שמרו סטטיסטיות: כמה בקשות, כמה הצלחו, כמה נכשלו

3.10 סיכום

בפרק זה למדנו את הבסיס הטכני לתקשורת עם שירותים מלאכותית. REST API הוא לא רק מושג טכני - הוא הגשר בין החזון העסקי שלכם לבין המימוש בפועל. הבנת JSON, קודי תגובה, Rate Limiting ואבטחת מפתחות היא הבדל בין פרויקט שמאлич לבין זהה שקורס תחת עומס או דולף נתונים.

3.10.1 נקודות מפתח לזכור

□ **REST API** = פרוטוקול תקשורת מבוסנה בין מערכות

□ **JSON** = פורמט חילופי נתונים קריא וგמיש

□ **קודם תגובה** = שפה משותפת להבנת מה קרה (200, 400, 401, 429, 500)

□ **Rate Limiting** = מגבלות ש策ריך לתוכנן להן מראש

□ **API Keys** = נכסים קרייטיים ש策ריכים ניהול ובטחה

□ **עלויות** = לא רק מחיר ה-API, אלא גם תשתיית, פיתוח ותחזוקה

3.10.2 מעבר לפרק הבא

REST API הוא הסטנדרט הנפוץ, אך האקויסיטיטם מתפתח. בפרק הבא נכיר את Model Context Protocol (MCP) - פרוטוקול חדש שנועד במפורש להעברת הקשר עשיר למודול שפה. נבין מתי להשתמש ב-MCP, מתי ב-REST, ואייך לשלב ביניהם. אבל קודם - תרגלו. בצעו את התרגילים, שחקו עם APIs, תעשו טעויות. זו הדרך היחידה ללמידה באמצעותם.