

פרק 3

תקשורת עם המכונה - sIPA TSER ויסודות NOSJ

מטרות הלמידה

בפרק זה נלמד:

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

3.1 פרולוג: שיחה שלא הובנה

דמיינו מנהל פרויקטים בחברת סטארטאפ, יושב בפגישה עם הצוות הטכני. המפתחת הראשית מציגה תוכנית לשילוב מודל שפה במערכת תמיכת הלקוחות. היא מדברת על "POST requests", על "JSON payloads", על "401 errors" ו-"rate limits". המנהל מהנהן בראשו, אך במוחו מתנהל דיאלוג פנימי שונה לגמרי: "אני מבין שזה חשוב, אבל מה זה בעצם אומר? איך זה משפיע על העלויות? על הזמן? על הסיכונים?"

זהו התסכול של מנהלים רבים בעידן הבינה המלאכותית. הטכנולוגיה מבטיחה, אך השפה שבה היא מתוארת נותרת זרה. הפרק הזה בא לגשר על הפער הזה. לא נהפוך אתכם למתכנתים, 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-IPA TSER

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

3.2.1 מהו IPA? אנלוגיה למלצר במסעדה

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

1. **מקבל את הזמנתכם** - הבקשה שלכם (Request)
 2. **בודק שהזמנה תקינה** - 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 משהו השתבש** - "המטבח עמוס", "המנה אזלה", "כרטיס האשראי נדחה"
- ב-REST API, כל אינטראקציה עוקבת אחר המבנה הזה: אתם שולחים בקשה מובנית, והשירות מחזיר תגובה מובנית.

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

REST מבוסס על ארבעה פעלים (HTTP Methods) המייצגים פעולות שונות:

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

טבלה 3.1: ארבעת פעלי ה-HTTP ושימושיהם

דוגמה מעשית: נניח שאתם בונים צ'אטבוט לתמיכת לקוחות. כל פעם שלקוח שולח הודעה:

1. המערכת שלכם תבצע GET לשלוח את הקשר השיחה הקודמת

2. תבצע POST לשלוח את השאלה החדשה ל-OpenAI

3. OpenAI יחזיר תשובה

4. המערכת תבצע POST נוסף לשמור את התשובה במאגר

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

3.2.3 מבנה הבקשה: noitacitnehtuA ,ydoB ,sredaeH

בקשת API היא כמו מעטפה בדואר:

□ **URL (Endpoint)** - הכתובת. לאן הבקשה הולכת?

`https://api.openai.com/v1/chat/completions`

□ **Headers** - מטא-מידע על הבקשה:

□ **Content-Type: application/json** - הנתונים בפורמט JSON

□ **Authorization: Bearer sk-...** - מפתח האימות שלכם

□ **User-Agent: MyCompany/1.0** - מזהה של המערכת שלכם

□ **Body** - הנתונים עצמם, בפורמט JSON:

```
{
  "model": "gpt-4",
  "messages": [
    {
      "role": "user",
      "content": "סכלש תורישל תישדוזה תולע יהח"
    }
  ],
  "temperature": 0.7
}
```

השלכה עסקית קריטית: כל Header יכול להשפיע על התנהגות השירות. למשל, אם תשכחו לשלוח Authorization, תקבלו שגיאת 401 Unauthorized. אם תגדירו Content-Type שגוי, השירות לא יבין את הנתונים ויחזיר 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:

□ **קריא 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 מיותרים (בניגוד ל-XML)

3.3.2 סוגי נתונים ב-NOSJ

JSON תומך בשישה סוגי נתונים בסיסיים:

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

טבלה 3.2: סוגי הנתונים ב-JSON

3.3.3 מבנים מקוננים: הכוח האמיתי של NOSJ

היכולת לקנן אובייקטים ומערכים היא מה שהופך את JSON לעוצמתי. הנה דוגמה למבנה ריאלי:

```
{
  "customer": {
    "id": "C12345",
    "name": "מ"עב תויגולונכט תרבח",
    "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"
}
```

שימו לב למבנה המקונן:

□ customer הוא האובייקט הראשי

□ subscription הוא אובייקט מקונן בתוך customer

□ usage_history הוא מערך של אובייקטים, כל אחד מייצג חודש

מבט מנהלי: כשאתם מבקשים מצוות הפיתוח "לשלוח את נתוני השימוש של הלקוח", המערכת תצטרך לנווט דרך המבנה הזה. אם תבקשו "עלות פברואר", היא תצטרך:

1. לגשת ל-customer

2. לחפש במערך usage_history

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

4. לחלץ את cost

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

3.3.4 תרגול: קריאת NOSJ מתועד IPA

הנה קטע אמיתי מתיעד OpenAI API לבקשת chat completion:

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

שאלות הבנה למנהלים:

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

2. כמה טוקנים נצרכו בסך הכל? (75)

3. איך תחשבו את העלות אם מחיר ה-input הוא \$0.01 לאלף טוקנים וה-output \$0.03 לאלף?

$$= (25 \times 0.01/1000) + (50 \times 0.03/1000) \\ = 0.00025 + 0.0015 = \$0.00175$$

זו החשיבה שאתם צריכים לאמץ: כל בקשה היא עלות. כל טוקן נספר. כשאתם מתכננים מערכת עם מיליון שיחות חודשיות, הבנת המבנה הזה קריטית.

3.4 קודי תגובה: מה המכונה אומרת לנו?

כל בקשת API מסתיימת בקוד תגובה (HTTP Status Code) - מספר תלת-ספרתי שמסכם מה קרה. הבנת הקודים הללו היא הבדל בין מנהל שידוע לשאול שאלות נכונות לבין מי שמנהל בפגישות טכניות.

3.4.1 מפת הקודים - ומה הם אומרים עליכם

קוד	משמעות טכנית	משמעות עסקית
200 OK	הבקשה הצליחה	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 תוכנית.
500 Internal Server Error	שגיאה בשרת	הבעיה בצד השירות, לא שלכם. צור קשר עם התמיכה.
503 Service Unavailable	שירות לא זמין	השרתים עמוסים או בתחזוקה. נסו שוב מאוחר יותר.

טבלה 3.3: קודי תגובה נפוצים והשלכותיהם

3.4.2 סיפור מהשטח: מקרה 924

חברת SaaS בינונית בישראל שילבה ChatGPT במערכת התמיכה שלה. בהשקה ראשונה, הכל עבד מצוין. יום למחרת, בשעה 00:09 בבוקר - קריסה מוחלטת. הצ'אטבוט חזר עם הודעות שגיאה. המנהלת הטכנית התקשרה לשיבת חירום.

הבעיה? קוד 429. התוכנית שלהם ב-OpenAI הייתה מוגבלת ל-60 requests per minute. בבוקר, כשכל נציגי השירות נכנסו למערכת בו-זמנית ושלחו בקשות, המערכת חצתה את המגבלה תוך שניות.

הלקח המנהלי:

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

3.5 gnitimiL etaR - מנהלים ומגבלות

3.5.1 מהו gnitimiL etaR ומדוע הוא קיים?

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

□ OpenAI Free Tier: 3 בקשות לדקה

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

□ Anthropic Claude: 50 בקשות לדקה ברמה בסיסית

למה? כי אחרת:

1. לקוח אחד יכול "לחנוק" את השרתים לכולם

2. עלויות החומרה יתפוצצו

3. קשה לחזות עומסים ולתכנן תשתית

3.5.2 חישוב tpuhguorhT מקסימלי

נניח שיש לכם מגבלה של 60 requests per minute, וכל בקשה לוקחת בממוצע 2 שניות (כולל זמן רשת ועיבוד). מהו ה-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 האמיתי כאן, לא ה-Rate Limit. אם הייתם שולחים את כל 60 הבקשות בפרץ בתחילת הדקה, היו נגמרות תוך שניה, ואז הייתם מחכים 59 שניות לדקה הבאה. לא יעיל.

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

3.5.3 תכנון ארכיטקטורת gnetimIL etaR

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

1. מהו נפח הבקשות הצפוי?

- ממוצע יומי? שעת שיא?
- דוגמה: 10,000 שיחות ביום = 6,944 שיחות ביום עסקי (16 שעות) = 7.2 שיחות לדקה בממוצע

2. האם המגבלה מספיקה?

- אם התוכנית מאפשרת 60 RPM ואתם צריכים רק 7.2 בממוצע - מצוין
- אבל מה בשעת שיא? אולי 30 RPM? עדיין בטוח

3. מה קורה כשחוצים את המגבלה?

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

4. מה העלות של שדרוג?

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

3.6 syeK IPA ואבטחה - מי שומר על השומרים?

3.6.1 מהו yeK IPA?

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

sk-proj-abc123def456ghi789jkl012mno345pqr678stu901vwx234

מפתח זה הוא למעשה הסיסמה לחשבון שלכם. מי שמחזיק בו יכול:

- לשלוח בקשות בשמכם
- לצרוך את המכסה שלכם
- לגרום לחיובים בכרטיס האשראי שלכם
- בתרחיש הגרוע - לגשת לנתונים רגישים אם המפתח מאפשר זאת

3.6.2 סיפור אימה: דליפת yeK IPA ב-buHtiG

סטודנט לתואר 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 והתחילו להשתמש בו.

עד שהסטודנט שם לב למחרת בבוקר, נצרכו \$1,200 בחשבון שלו. OpenAI לא החזירו את הכסף - זו אחריותו של המשתמש לשמור על המפתח. לקח: API Keys הם כמו כרטיסי אשראי. לעולם לא מפרסמים אותם.

3.6.3 מדיניות ניהול syeK IPA בארגון

כמנהלים, עליכם לוודא שהארגון מיישם את העקרונות הבאים:

1. ניהול סודות (Secrets Management)

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

2. עקרון ההרשאה המינימלית (Least Privilege)

- צרו מפתחות שונים למטרות שונות
- דוגמה: מפתח לפיתוח (מגבלה נמוכה), מפתח לייצור (מגבלה גבוהה)
- הגבילו הרשאות - מפתח לקריאה בלבד sv קריאה+כתיבה

3. רוטציה תקופתית

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

4. ניטור ואזעקות

- התראה על שימוש חריג (פתאום 1,000 בקשות לדקה)
- התראה על חריגה מתקציב (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 ומדדים עסקיים

3.7.1 נוסחת עלות לבקשה

כל בקשה ל-API של שירות LLM עולה כסף. 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 העלות הישירה. העלות 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 פנייה:

□ Input: 200 טוקנים (הקשר + שאלת הלקוח)

□ Output: 150 טוקנים (תשובת המודל)

□ 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

□ שרת (AWS): \$200/חודש

□ ניטור וכלים: \$100/חודש

□ סה"כ: \$3,750/חודש

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

3.7.2 תכנון תקציב IPA

כמנהלים, עליכם לשאול:

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

דוגמת מדיניות:

□ תקציב חודשי: \$1,000

□ התראה ב-70%: הודעה למנהל

□ התראה ב-90%: הפסקת שירותים לא קריטיים

□ ב-100%: חסימה מוחלטת (למנוע חריגה)

3.8 דוגמאות מעשיות מעולם העסקים

3.8.1 דוגמה 1: חיבור אפליקציה פנימית ל-IPA IAnepO

תרחיש: חברת SaaS רוצה להוסיף תכונת "סיכום חכם" למערכת CRM שלהם. כל פגישה עם לקוח תתועד, ובלחיצת כפתור המערכת תייצר סיכום. **שלבי האינטגרציה:**

1. רכישת מפתח API

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

2. פיתוח הלוגיקה

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

3. טיפול בשגיאות

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

4. ניטור

- לוג של כל בקשה - מתי, מי, כמה טוקנים, כמה עלה
- דאשבורד חודשי: סה"כ בקשות, עלויות, זמני 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: $1400\% = (3000-200)/200$ תשואה חודשית!

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

תרחיש: צוות מכירות רוצה "עוזר חכם" שיכול לענות על שאלות כמו "מי הלקוחות שלא קנו מאיתנו 6 חודשים ושויים מעל \$50K?"
ארכיטקטורה:

1. שלב 1: שאילתה ל-CRM API

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

2. שלב 2: העברה ל-LLM

- הכנת 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
- כפתורי פעולה: "שלח מייל", "קבע פגישה"

אתגרים שנתקלו בהם:

- **Rate Limiting של Salesforce:** 1,000 בקשות ליום. פתרון: קאש יומי של נתונים
- **גודל ההקשר:** רשימה 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: דאשבורד בזמן אמת של נתוני IA

תרחיש: מנהל טכנולוגי רוצה לראות "מה קורה עכשיו" במערכת ה-AI.
מדדים בדאשבורד:

- **בקשות לדקה** - האם מתקרבים ל-Rate Limit?
- **עלות שעתית** - כמה הוצאנו עד עכשיו היום?
- **זמני תגובה** - האם יש האטה?
- **קודי שגיאה** - כמה 429, 500, וכו'?
- **טוקנים ממוצעים** - האם המשתמשים שולחים שאלות ארוכות מדי?

יישום טכני:

- כל בקשה נשמרת ב-Database עם timestamp
- Python script מריץ אגרגציות כל 5 דקות
- ממשק Streamlit מציג את הנתונים בזמן אמת

ערך עסקי:

- זיהוי בעיות לפני שהלקוחות 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 תיאורטיים

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 $\text{Input} = \$15/\text{million tokens}$, $\text{Output} = \$75/\text{million tokens}$

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

תרגיל 2.3. ניתוח קודי שגיאה וכתובת נהלי טיפול

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

□ 401 Unauthorized - 15 מקרים

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

□ 500 Internal Server Error - 3 מקרים

□ 400 Bad Request - 8 מקרים

משימה:

1. לכל קוד שגיאה, הציעו סיבה אפשרית

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

3. הציעו שינוי ארכיטקטוני למניעת ה-429 בשעות השיא

תרגיל 3.3. תכנון ארכיטקטורת gnetimil etaR

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

□ 30% מהעובדים ישתמשו יום-יום

□ כל משתמש ישלח בממוצע 10 שאלות ביום

□ יום עבודה: 8 שעות

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

התוכנית הנוכחית: 60 requests/min.

משימה:

1. חשבו את ממוצע הבקשות לדקה ביום רגיל

2. חשבו את שיא הבקשות בשעת העומס

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

4. הציעו 3 אסטרטגיות להתמודדות ללא שדרוג מיד

תרגיל 4.3. כתיבת מדיניות ניהול syeK IPA

אתם סמנכ"ל טכנולוגיה של חברה שמשתמשת ב-5 שירותי API שונים: OpenAI, AWS, Twilio, SendGrid, Stripe.

משימה: כתבו מדיניות ארגונית בת 2 עמודים שמכסה:

1. מי מורשה ליצור מפתחות?

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

3. כמה זמן מפתח תקף?

4. מה קורה כשעובד עוזב?

5. מה התהליך אם מפתח דלף?

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 תמיכה

□ 200 שיחות ביום

□ כל שיחה: 500 טוקנים input, 200 טוקנים output

תרחיש C: ניתוח חוזים

□ 10 חוזים ביום

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

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

משימה:

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

2. דרגו מהיקר לזול

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

4. הציעו אופטימיזציה לתרחיש היקר ביותר

3.9.2 תרגילי קוד nohtyP

תרגיל 6.3. שליחת בקשה ל-IPA IAnepO 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: דוקה תא ומילשה
```

דרישות:

□ השתמשו בספריות: requests, python-dotenv

□ הקוד צריך להיות ברור וקריא

□ הוסיפו הערות מסבירות

תרגיל 7.3. ניהול cigoL yrteR עם gnitimiL etaR

כתבו מחלקה APIClient שמטפלת אוטומטית ב-Rate Limiting:

1. אם מתקבל 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 מושיי
        pass

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

בנוס:

□ הוסיפו exponential backoff (המתנה גדלה עם כל ניסיון)

□ שמרו סטטיסטיקות: כמה בקשות, כמה הצליחו, כמה נכשלו

3.10 סיכום

בפרק זה למדנו את הבסיס הטכני לתקשורת עם שירותי בינה 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 שמצליח לבין כזה שקורס תחת עומס או דולף נתונים.

3.10.1 נקודות מפתח לזכור

- REST API = פרוטוקול תקשורת מובנה בין מערכות
- JSON = פורמט חילופי נתונים קריא וגמיש
- קודי תגובה = שפה משותפת להבנת מה קרה (200, 400, 401, 429, 500)
- Rate Limiting = מגבלות שצריך לתכנן להן מראש
- API Keys = נכסים קריטיים שצריכים ניהול ואבטחה
- עלויות = לא רק מחיר ה-API, אלא גם תשתית, פיתוח ותחזוקה

3.10.2 מעבר לפרק הבא

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