

מבחן שרתים – שרת הצפנות

עליך לממש שרת FastAPI המספק שירותי הצפנה ופענוח שונים. הצפנים פועלים על טקסט באנגלית, וללא רגישות לאותיות גדולות או קטנות (למשל - אפשר להחזיר תמיד אותיות קטנות, כל עוד התשובה עומדת באלגוריתם ההצפנה גם אם התקבלה בקשה באותיות גדולות) כמשימת בונוס יש גם ניהול סטטיסטיקות שקשורות לביצועים של השרת.

(1) מטרות

- בניית שרת REST בעזרת FastAPI.
- מימוש של ENDPOINTS שמייצגים 2 סוגי צופן (ראו נספח א' לפירוט ודוגמאות לכל צופן):
 - קיסר (Caesar) – הצפנה ופענוח עם פרמטר היסט (offset).
 - גדר (Fence) – הצפנה ופענוח לפי מיקומים זוגיים ואי-זוגיים.
- מימוש מידלור זמן תגובה (בונוס).
- שמירת סטטיסטיקות שימוש בקבצי JSON (בונוס).

(2) דרישות מקדימות

- יש ליצור REPO חדש ולעשות לו CLONE.
- צרו קובץ README.md שבו השם המלא שלכם, הכיתה ות"ז - כולם בשורה נפרדת כל אחד.
- צרו סביבה וריטואלית בפייתון (uv / poetry / venv וכד'). הפעילו אותה וודאו שאתם בתוכה.
- התקינו בתוך הסביבה את הספריות: fastapi pydantic בעזרת pip. אתם יכולים גם להתקין את uvicorn אם תרצו להריץ את השרת בעזרתה. אפשר להתקין ספריות נוספות אם אתם רוצים.

- חובה ליצור קובץ requirements.txt בעזרת pip freeze. ודאו כי אחרי הרצת הפקודה נוצר קובץ בו אתם רואים את כל dependencies של הפרוייקט.
- הרצת השרת תיעשה בעזרת הפקודה: fastapi dev main.py או:

uvicorn main:app --reload

(3) דרישות כלליות

1. כל התשובות מכל הENDPOINTS ייעשו בפורמט JSON.
2. יש להוסיף PRINT ותיעוד לכל פעולה משמעותית שמתבצעת, ויש לוודא לפני תום המבחן שלא השארתם הדפסות שאינן כאלה.
3. רצוי מאוד לוודא את תקינות הקוד של השרת בעזרת קריאות עם CURL. בנספח ג' יש דוגמאות לטקסטים מוצפנים שאפשר להיעזר בהם כדי לוודא שתהליך הפיענוח עובד.

(4) מבנה הפרויקט (מומלץ בלבד)

חובה לממש את בסיס השרת בקובץ הראשי [main.py](#). אפשר ואף מומלץ לחלק לראוטרים (אם אתם יודעים איך) ופונקציות עזר.

(5) קובץ README (חובה)

יש ליצור קובץ בשם README המכיל שלוש שורות בלבד:

שם מלא בעברית

כיתה

תעודת זהות

(6) הגדרת ה-Endpoints (ראו הגדרת צפנים בנספח א')

basic endpoints:

GET /test

Response: { "msg": "hi from test" }

תיאור: נקודת קצה בסיסית לצורך בדיקות.

GET /test/:name

Response: { "msg": "saved user" }

תיאור: חילוץ השם מתוך ה-PATH PARAMS ושמירתו בתוך קובץ בשם names.txt.
אין לדרוס טקסט קיים - רק להוסיף.

Caesar cipher endpoint:

POST /caesar

Body: { "text": string, "offset": int, "mode": "encrypt"/"decrypt" }

Response: { "encrypted_text": "..." } או { "decrypted_text": "..." }

תיאור:

מצפין או מפענח טקסט לפי צופן קיסר עם היסט (offset) מוגדר.

Fence cipher endpoints:

GET /fence/encrypt?text=<טקסט>

Response:

{ "encrypted_text": "..." }

תיאור - מחלץ את הטקסט מהURL ומצפין אותו לפי צופן הגדר

POST /fence/decrypt

Body: { "text": "string" }

Response: { "decrypted": "..." }

תיאור - מקבל את הטקסט מהגוף של הבקשה ומפענח אותו לפי צופן גדר.

בונוס: שמירת נתוני ביצועים

לצד ה-Endpoints, השרת נדרש לתעד נתונים על כל בקשה שמתקבלת ומטופלת, לצורך מעקב אחר ביצועי המערכת. המידע נשמר בשני קבצים בתוך תיקיית /data:

1. endpoints_data.json

קובץ זה שומר מידע מצטבר לכל אנדפוינט. הפורמט הוא רשימה של מילונים, כאשר כל מילון מייצג אנדפוינט יחיד/ במילון נשמר השם של הENDPOINT והמתודה, וסטטיסטיקה: total_requests_received - כמות הבקשות שהתקבלו לאותו ENDPOINT. avg_handling_time - זמן ממוצע מרגע קבלת הבקשה ועד שליחת התשובה.

מבנה קובץ לדוגמה:

```
[
  {
    "url": "/atbash",
    "method": "POST",
    "stats": {
      "total_requests_received": 12,
      "avg_handling_time": 10.4
    }
  }
]
```

- בעת קבלת בקשה: יש לעדכן את total_requests_received.

- בעת שליחת תגובה: יש לחשב ולעדכן את avg_handling_time.
- בעת שליחת תגובה, אחרי עדכון avg_handling_time, יש לעדכן את summary.json, כמתואר בסעיף הבא.

2. summary.json

קובץ זה מרכז את הנתונים מכל האנדפוינטים, ומציג ערכי שיא ומינימום לפי סוג מדידה:

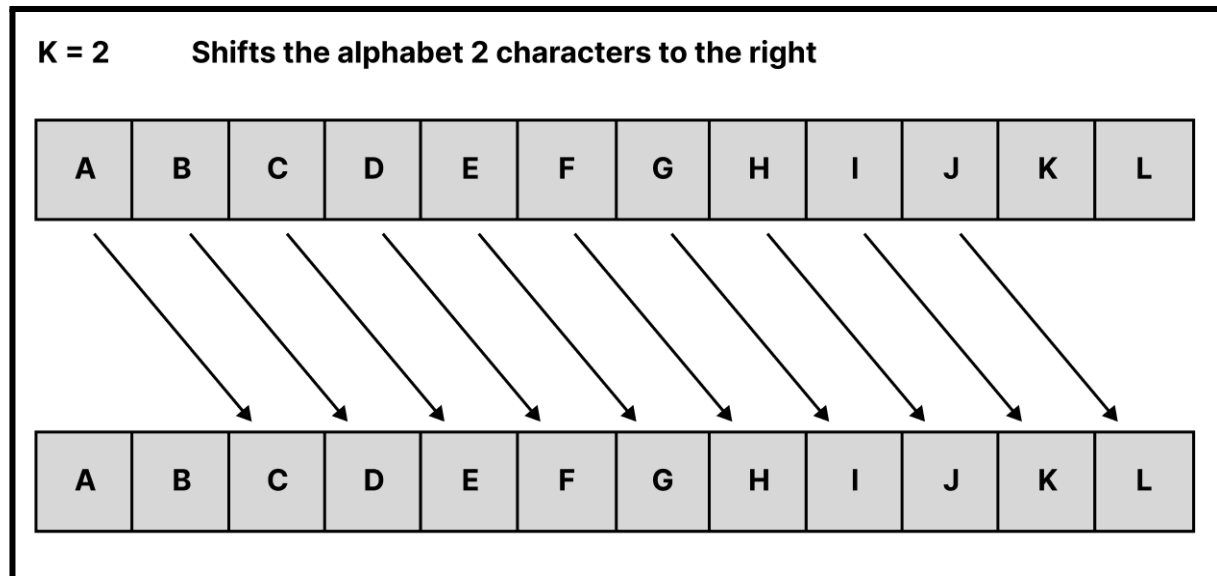
```
{
  "highest_requests": {
    "name": "atbash POST",
    "number": 17
  },
  "lowest_requests": {
    "name": "fence GET",
    "number": 3
  },
  "highest_handling_time": {
    "name": "caesar POST",
    "number": 21.7
  },
  "lowest": {
    "name": "atbash POST",
    "number": 7.9
  }
}
```

יש להוסיף ENDPOINT שמקבל את הנתונים הכללים, ונתונים לפי URL וMETHOD.

צופן קיסר

צופן קיסר הוא צופן החלפה מבוסס היסט קבוע באלפבית. כל אות מוזזת במספר צעדים קבוע (offset) קדימה. כאשר מגיעים לסוף האלפבית, ממשיכים מהתחלה.

לדוגמה, בהיסט 2:



אם מגיעים לסוף האלפבית, חוזרים להתחלה. לדוגמה, אם צריך להצפין את Z עם offset של 2: חוזרים ל A וממשיכים ל B.

הפענוח זהה להתהליך ההצפנה - רק בכיוון ההפוך.

צופן גדר

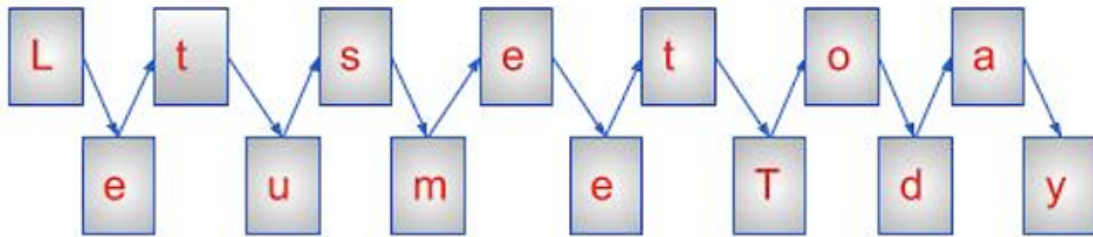
צופן גדר (או Rail Fence Cipher) הוא צופן טרנספוזיציה (החלפת סדר האותיות). הטקסט נכתב לסירוגין בשתי "מסילות" – אחת לאותיות במיקומים אי-זוגיים והשנייה לזוגיים, ואז מוצמדות שתי המסילות זו לזו.

בגרסה שלנו משתמשים ב-2 מסילות בלבד. אפשר להשתמש ביותר.

תהליך הצפנה

1. מסירים רווחים.
2. מפרידים את האותיות למיקומים אי-זוגיים וזוגיים. האות הראשונה באינדקס 0 נחשבת זוגית.
3. מצמידים: תחילה האותיות הזוגיות ולאחר מכן האי זוגיות.

לדוגמה: הטקסט: let us meet today:



הטקסט המוצפן: ltssetoaeumetdy

תהליך פענוח (אין צורך לשחזר את הרווחים):

תחשבו.

נספח ב – אריתמטיקת שעון (Clock Arithmetic / Modulo)

הסבר כללי

אריתמטיקת שעון היא שיטה מתמטית שבה המספרים "נעים במעגל", וכשמגיעים לסוף — חוזרים להתחלה. השיטה מבוססת על פעולת **מודולו (mod)**, %, המגדירה את שארית החלוקה של מספר אחד במספר אחר.

בדומה לשעון עם הספרות 0-11, שבו $9 + 4$ לא שווה 13 אלא 1:



כך גם באלפבית — אחרי האות האחרונה חוזרים לראשונה. העיקרון הזה הוא הבסיס לפעולת צופן קיסר.

דוגמה – על שעון עם 12 ספרות.

נניח שהשעה עכשיו 9:00,
ואנו מוסיפים 5 שעות קדימה.
נחשב:

$$(9 + 5) \bmod 12 = 2$$

$$(9 + 5) \% 12 = 2 \text{ \# in python}$$

כלומר, התוצאה היא השעה 2:00.

דוגמה – באלפבית האנגלי

נניח ש-A היא 0, B היא 1, ... Z היא 25.
אם נרצה להזיז את האות W (אינדקס 22) ב-5 צעדים קדימה:

$$(22 + 5) \bmod 26 = 1$$

$$(22 + 5) \% 26 = 1 \text{ \# in python}$$

1 מייצג את האות B.
כלומר, W עם offset של 5 הופכת ל-B.

נספח ג' - דוגמאות לקוד מקודד ומפוענח, בכל צופן

הטקסטים הבאים הם לשימושכם, כחלק מקריאות CURL לשרת - על מנת לוודא שה-ENDPOINT באמת עובד.

צופן קיסר:

טקסט	הסטה	תוצאה
hello world	7	olssv dvysk
mubsecu aetaet	16	welcome kodkod

צופן גדר:

טקסט	תוצאה
hi there	hteeihr
wloeokdecmkdo	welcomekodkod

נספח ד' - פקודת CURL מוכנות לבדיקה:

🧩 1. בדיקת תקינות שרת

```
curl -i -X GET "http://127.0.0.1:8000/test"
```

Expected output:

```
{  
  "msg": "hi from test"  
}
```

🧩 2. שמירת שם ל- names.txt

```
curl -i -X GET "http://127.0.0.1:8000/test/Avi"
```

Expected output:

```
{  
  "msg": "saved user"  
}
```

(*"Avi" תתווסף השורה names.txt בקובץ*)

🔒 3. Caesar encrypt — hello world, offset 7

```
curl -i -X POST "http://127.0.0.1:8000/caesar" -H "Content-Type:  
application/json" -d '{"text":"hello  
world","offset":7,"mode":"encrypt"}'
```

Expected output:

```
{  
  "encrypted_text": "olssv dvysk"  
}
```

4. Caesar decrypt — olssv dvysk, offset 7

```
curl -i -X POST "http://127.0.0.1:8000/caesar" -H "Content-Type:  
application/json" -d '{"text":"olssv  
dvysk","offset":7,"mode":"decrypt"}'
```

Expected output:

```
{  
  "decrypted_text": "hello world"  
}
```

5. Caesar encrypt — welcome kodkod, offset 16

```
curl -i -X POST "http://127.0.0.1:8000/caesar" -H "Content-Type:  
application/json" -d '{"text":"welcome  
kodkod","offset":16,"mode":"encrypt"}'
```

Expected output:

```
{
```

```
"encrypted_text": "mubsecu aetaet"
}
```

6. Caesar decrypt — mubsecu aetaet, offset 16

```
curl -i -X POST "http://127.0.0.1:8000/caesar" -H "Content-Type:
application/json" -d '{"text":"mubsecu
aetaet","offset":16,"mode":"decrypt"}'
```

Expected output:

```
{
  "decrypted_text": "welcome kodkod"
}
```

7. Fence encrypt — hi there

```
curl -i -X GET "http://127.0.0.1:8000/fence/encrypt?text=hi%20there"
```

Expected output:

```
{
  "encrypted_text": "hteeihr"
}
```

8. Fence decrypt — hteeihr

```
curl -i -X POST "http://127.0.0.1:8000/fence/decrypt" -H "Content-Type: application/json" -d "{\"text\": \"hteeihr\"}"
```

Expected output:

```
{  
  "decrypted": "hithere"  
}
```

9. Fence encrypt — welcome kodkod

```
curl -i -X GET  
"http://127.0.0.1:8000/fence/encrypt?text=welcome%20kodkod"
```

Expected output:

```
{  
  "encrypted_text": "wloeokdecmkdo"  
}
```

10. Fence decrypt — wloeokdecmkdo

```
curl -i -X POST "http://127.0.0.1:8000/fence/decrypt" -H "Content-Type: application/json" -d "{\"text\": \"wloeokdecmkdo\"}"
```

Expected output:

```
{  
  "decrypted": "welcomekodkod"  
}
```