

פרויקט קורס: שרת מטרות צבאיות (Military Targets Server)

תיאור כללי

בפרויקט זה תבנו שרת Express.js המדמה מערכת לניהול מטרות צבאיות. השרת מאפשר יצירה, קריאה, עדכון ומחיקה (CRUD) של מטרות, תוך עבודה הדרגתית עם:

- HTTP ו-REST
- Headers, Path Params, Query Params, Body
- Middleware
- עבודה עם JSON
- Node.js File IO
- יציבות, תיעוד ודיפלויימנט

⚠ **עיקרון קריטי לפרויקט כולו**
בכל הפאזות:

- כל הנתונים נקראים ונכתבים **מקבצי JSON**
- אין עבודה עם DB
- אין "שמירה בזיכרון בלבד"
- כל endpoint שעובד עם מטרות → משתמש ב-`fs` Node.js

מבנה נתונים — Target (מטרה צבאית)

כל מטרה מיוצגת כאובייקט JSON.

```
{
  "id": "t-102",
  "codeName": "Iron Falcon",
  "region": "north",
  "priority": 4,
  "status": "new",
  "createdAt": "2025-12-21T09:30:00.000Z"
}
```

פירוט שדות

שדה	תיאור
id	מזהה ייחודי של המטרה
codeName	שם קוד של המטרה
region	אזור פעילות: north, south, center, unknown
priority	רמת חשיבות 1-5 (5 = הגבוהה ביותר)
status	new, approved, in_progress, completed
createdAt	זמן יצירת המטרה (ISO)

מבנה קבצים מחייב

/data
targets.json — L

```
}
  "targets": []
}
```

פאזה 1 — HTTP בסיסי וקריאה מקבצים

(READ ONLY)

מטרת הפאזה

להבין:

- איך שרת מקבל מידע מהלקוח
- באילו דרכים שונות מעבירים מידע בבקשת HTTP
- למה Header ≠ Path ≠ Query
- איך לקרוא נתונים מקובץ JSON ולהחזיר תשובה

⚠ בפאזה זו:

- משתמשים רק ב-GET
- לא משנים קבצים

- כל נתון נקרא מ-`targets.json`

תרגיל 1 — GET /health

בדיקת תקינות שרת

למה זה קיים?

זה endpoint בסיסי לניטור שרתים.
מאפשר לבדוק שהשרת מאזין ופועל.

תגובה

```
}
  "status": "ok"
  "serverTime": "ISO_TIMESTAMP"
{
```

תרגיל 2 — GET /briefing

זיהוי הלקוח באמצעות Header

למה Header?

זה מידע שמזהה את השולח, לא את המשאב.

Header חובה

Client-Unit: Golani

אם חסר → 400

תגובה

```
}
  "unit": "Golani"
  "message": "briefing delivered"
{
```

תרגיל 3 — GET /targets/:id

שליפת מטרה לפי מזהה

למה Path Param?

כי ה-ID הוא חלק מהמשאב עצמו.

מה השרת עושה

- קורא את `targets.json`
- מחפש מטרה עם ID תואם
- מחזיר 404 או

תרגיל 4 — GET /targets

שליפת אוסף מטרות + סינון

Query Params

- region
- status
- minPriority

למה Query?

כי אלו פילטרים על אוסף, לא משאב אחד.

+ אקסטרה פאזה 1

GET /targets/:id/brief

תצוגה מצומצמת של מטרה (לא כל הנתונים)

GET /intel/ping

Header לזיהוי היחידה + Query לרמת פירוט

GET /targets/search

חיפוש חופשי לפי `q` (לא התאמה מדויקת)

פאזה 2 — Middleware, JSON Body וכתובה לקבצים

מטרת הפאזה

להבין:

- ש-endpoint לא אמור להכיל הכל
- למה יש Middleware
- איך שרת מקבל נתונים מורכבים
- איך שינויים נשמרים לקובץ

תרגיל 1 — Middleware לוג

למה?

כדי להימנע מכפילויות ולרכז לוגיקה חוזרת.

מדפיס:

- method
- path
- timestamp

תרגיל 2 — Middleware שמוסיף Header

מוסיף לכל Response:

X-Server-Start-Time

למה Middleware?

זה מידע תשתיתי, לא עסקי.

תרגיל 3 — POST /targets

יצירת מטרה חדשה

למה Body?

כי יוצרים משאב עם כמה שדות.

Flow

1. בדיקת Content-Type
2. קריאת Body
3. ולידציה
4. קריאה מהקובץ
5. הוספה
6. כתיבה לקובץ

תרגיל 4 — PUT /targets/:id

עדכון מטרה

שילוב:

- Path Param (איזו מטרה)
- Body (מה משתנה)

+ אקסטרה פאזה 2

POST /targets/:id/status

עדכון נקודתי של סטטוס

PUT /targets/:id/priority

עדכון עדיפות בלבד

POST /debug/echo

החזרת כל נתוני הבקשה לצורכי דיבוג

פאזה 3 — CRUD מלא, יציבות ודיפלוימנט

מטרת הפאזה

לבנות שרת:

- יציב
- מתועד
- עובד אחרי restart
- מוכן לפרודקשן

תרגיל 1 — DELETE /targets/:id

מחיקת מטרה מהקובץ

תרגיל 2 — Error Handling מרכזי

למה קריטי?

- קריאה/כתיבה לקובץ עלולה להיכשל
- אסור לקרוס או לחשוף שגיאות פנימיות

תרגיל 3 — README

תיעוד:

- מבנה המטרה
- כל ה-endpoints
- דוגמאות שימוש

תרגיל 4 — Deployment

דרישות:

- PORT מתוך env
- יצירת קובץ אם חסר
- שרת עובד אחרי restart

3 + אקסטרה פאזה

DELETE /targets

מחיקה לפי פילטרים בלבד (הגנה מפרודקשן)

GET /intel/summary

סיכום סטטיסטי במקום raw data

POST /targets/bulk

יצירה מרובה + טיפול בשגיאות חלקיות

תוצר סופי

שרת Express.js מלא:

- CRUD
- File IO
- Middleware
- תיעוד
- רץ בפרודקשן

אם תרצה, השלב הבא יכול להיות:

- Rubric בדיקה מפורט לפי פאזה
- Checklist בדיקות לתלמיד
- גרסת מבחן עם דרישות מינימום
- Skeleton קוד התחלתי

פשוט תגיד.