

מבחן MongoDB

פרוייקט “מאגר העובדים”

Operation Silent Payroll

בסוף שנת 2025 התקבלו במוסד התרעות מגורמי מודיעין מערביים על חברת טכנולוגיה בינלאומית בשם **Helios Dynamics Ltd**.

החברה פועלת כחברת הייטק לגיטימית לכל דבר – פיתוח תוכנה, ייעוץ ו־Data Services – עם מאות עובדים, מחלקות מסודרות ומשכורות תואמות שוק.

עם זאת, ניתוח ראשוני העלה חשד כי חלק מהעובדים בחברה עשויים להיות **שתולים איראניים**, הפועלים תחת כיסוי אזרחי ומנהלים קשרים לוגיסטיים וכספיים עבור רשתות טרור בינלאומיות.

מאגר העובדים של החברה, הכולל נתונים על תפקידים, מחלקות, ותק ושכר, הועבר ליחידת הסייבר של המוסד לצורך תחקור ראשוני.

המשימה שלכם:

המשימה שלכם היא לבנות מערכת חקירתית שתנתח מאגר נתונים ארגוני של עובדים, במטרה:

- לאתר דפוסים חריגים
- לזהות קבוצות עובדים חשודות
- ולהפיק תובנות ראשוניות מתוך נתונים שנראים לכאורה שגרתיים

המערכת תבצע ניתוח על בסיס שאילתות ותציג את התוצאות דרך ממשק קיים.

המטרה אינה בהכרח להפליל עובדים מסויימים, אלא **לאפשר המשך חקירה** על-ידי הצפת מידע רלוונטי וחריג.

כל שאילתה שאתם מממשים מייצגת פעולה חקירתית לזיהוי תבניות ארגוניות חריגות. כלומר, כל תוצאה עשויה להיות קצה חוט.

טכנולוגיות ומימוש

המערכת תכלול את הרכיבים הבאים:

- שרת Backend מבוסס FastAPI
- מסד נתונים MongoDB (הרצה באמצעות Container)
- סביבת הרצה ופריסה:
- OpenShift (לסביבת ענן)

המערכת תאפשר תקשורת מלאה בין שרת האפליקציה למסד הנתונים, ותספק גישה לנתונים דרך נקודות קצה מוגדרות בשרת האפליקטיבי.

דרישות המערכת

עליכם לממש מערכת צד-שרת אשר:

- מתחברת למסד נתונים
- מבצעת שאילתות על מאגר הנתונים בהתאם למשימות החקירה
- מחזירה את תוצאות השאילתות בצורה מובנית בהתאם לדרישות

המערכת נדרשת:

- לאפשר שליפה וסינון של נתונים
- לתמוך בבדיקת תנאים והשוואות בין ערכים
- להחזיר תוצאות מלאות או חלקיות בהתאם לדרישה

כל פעולה במערכת מייצגת שלב בתהליך חקירה, וכל פלט משמש כבסיס לניתוח והסקת מסקנות.

עזרים מותרים ואיסורים

✗ איסורים (חובה לקרוא)

במהלך המבחן חל איסור מוחלט על:

- שימוש ב-Google או כל מנוע חיפוש אחר

- שימוש ב־כלי AI מכל סוג (כולל אך לא מוגבל ל־ChatGPT, Copilot, Claude, Gemini וכדומה)
 - קבלת עזרה מאדם אחר
-

עזרים מותרים

מותר להשתמש אך ורק במקורות הבאים:

Python

- תיעוד רשמי:

[Python Docs](#)

[PyMongo - Documentation](#)

MongoDB

- תיעוד רשמי:

[MongoDB Documentation - Homepage](#)

Kubernetes

- תיעוד רשמי:

[Kubernetes Documentation](#)

OpenShift

- תיעוד רשמי:

[Red Hat Documentation](#)

Docker

- תיעוד רשמי:

[Docker Docs](#)

- Docker Hub (Images, Tags, Usage):

[Docker Hub](#)

FastAPI

- תיעוד רשמי:

[FastAPI](#)

Git

- תיעוד רשמי:

[Git - Reference](#)

YAML

- YAML Lint (בדיקת תקינות קבצי YAML):

[The YAML Validator](#)

אתרי עזר כלליים

- W3Schools:

[W3 Schools](#)

- GeeksforGeeks:

[GeeksforGeeks](#)

- [/https://www.geeksforgeeks.org/mongodb/mongodb-query-and-projection-operator](https://www.geeksforgeeks.org/mongodb/mongodb-query-and-projection-operator)

מבנה תיקיות וקבצים

הפרויקט בנוי במבנה הבא:

```
project-root/
|
├─ app/
|   ├── main.py
|   ├── connection.py
|   ├── *optional: routes.py
|   └─ dal.py
|
├─ k8s/
|   └─ *.yaml
|
├─ Dockerfile
└─ requirements.txt
```

תיאור כללי:

- `main.py` – הגדרת ה-API, ונקודות הקצה של השרת
- `connection.py` – חיבור למסד הנתונים
- `dal.py` – לוגיקת שאילתות וגישה לנתונים
- `/k8s` – קבצי YAML לפריסה
- `Dockerfile` – בניית האפליקציה
- `requirements.txt` – תלויות הפרויקט
- `routes.py` - אופציונלי: קובץ נפרד להגדרת נקודות הקצה של השרת

קובץ `connection.py` – חיבור למסד הנתונים

בשלב זה עליכם לממש חיבור פעיל למסד הנתונים.

הקובץ `connection.py` אחראי על:

- יצירת חיבור למסד הנתונים
- הגדרת מסד נתונים (Database) ואוסף נתונים (Collection) עבור נתוני העובדים
- חשיפת החיבור לשימוש בקבצים אחרים בפרויקט

יצירת מסד הנתונים וה־Collection

יצירת מסד הנתונים וה־Collection היא לבחירתכם:

- ניתן ליצור אותם אוטומטית דרך הקוד
- ניתן להסתמך על יצירה אוטומטית של MongoDB בעת טעינת הנתונים

עם זאת, החיבור עצמו חייב להתבצע באמצעות קוד בלבד ולא באמצעות כלים חיצוניים בלבד.

הכנסת הנתונים למסד הנתונים

טעינת הנתונים למסד הנתונים היא חלק מהמשימה, אך אופן הטעינה נתון לבחירתכם.

מותר להשתמש בכל דרך נוחה:

- קוד Python
- טרמינל / Mongo Shell
- Docker / Docker Compose

דגשים חשובים

- החיבור למסד הנתונים חייב להיות ממומש בקוד
- אין לכתוב לוגיקה חקירתית בקובץ זה

קובץ **dal.py** – פונקציות נדרשות

שימו לב: את הלוגיקה של כלל הפונקציות יש לממש באמצעות שאילתות מול שירות MongoDB באמצעות חיבור פייתוני. אין לממש את השאילתות באמצעות לוגיקה פייתונית בלבד.

הערה: אין חובה להחזיר את השדה "_id" בשאלות בהן אתם נדרשים להחזיר את כל הפרטים של העובד.

get_engineering_high_salary_employees

שאלה 1 – עובדים עם שכר גבוה במחלקת Engineering

כתבו שאילתה שמחזירה את פרטי העובדים הבאים:

- `employee_id`
- `name`
- `salary`

עבור עובדים אשר:

- עובדים במחלקת **Engineering**
- והשכר שלהם גבוה מ-**65,000**

get_employees_by_age_and_role

שאלה 2 – עובדים לפי גיל ותפקיד

כתבו שאילתה שמחזירה את כל פרטי העובדים אשר:

- גילם בין **30 ל-45** (כולל את שני הקצוות)
 - והתפקיד שלהם הוא **Engineer** או **Specialist**
-

get_top_seniority_employees_excluding_hr

שאלה 3 – עובדים עם ותק גבוה (ללא HR)

כתבו שאילתה שמחזירה את כל פרטי העובדים כאשר:

- אלו 7 העובדים עם הוותק הגבוה ביותר
- הם אינם עובדים במחלקת HR

get_employees_by_age_or_seniority

שאלה 4 – עובדים לפי גיל או ותק

כתבו שאילתה שמחזירה את השדות:

- employee_id
- name
- age
- years_at_company

עבור עובדים אשר:

- גילם גדול מ-50

או

- הוותק שלהם בחברה קטן מ-3 שנים
-

get_managers_excluding_departments

שאלה 5 – מנהלים שאינם במחלקות מסוימות

כתבו שאילתה שמחזירה את כל פרטי העובדים אשר:

- התפקיד שלהם הוא **Manager**
 - אך הם אינם עובדים באף אחת מהמחלקות:
 - Sales
 - Marketing
-

get_employees_by_lastname_and_age

שאלה 6 – עובדים לפי שם משפחה וגיל

כתבו שאילתה שמחזירה את:

- `name`
- `age`
- `job_role.department`

עבור עובדים אשר:

- שם המשפחה שלהם הוא **Nelson** או **Wright** (תשומת לב לכך שבנתונים מופיע השם המלא, ועל כן הלוגיקה צריכה להיות בהתאם)
 - וגילם קטן מ-35
-

קובץ `main.py` – REST API Endpoints

עליכם לממש שרת REST API אשר חושף את נקודות הקצה הבאות.
כל נקודת קצה מבצעת קריאה לפונקציה המתאימה בקובץ `dal.py` ומחזירה JSON תקני.

כל ה-Endpoints:

- משתמשים בבקשת **GET**
אינם מקבלים Body
- מחזירים נתונים בלבד

1. עובדים עם שכר גבוה במחלקת Engineering

:Route

`GET /employees/engineering/high-salary`

מחזיר עובדים ממחלקת Engineering עם שכר גבוה מהסף שנקבע.

2. עובדים לפי גיל ותפקיד

:Route

`GET /employees/by-age-and-role`

מחזיר עובדים בהתאם לטווח גיל ותפקידי עבודה מוגדרים.

3. עובדים עם ותק גבוה (ללא HR)

:Route

GET /employees/top-seniority

מחזיר את העובדים בעלי הוותק הגבוה ביותר, תוך סינון מחלקת HR.

4. עובדים לפי גיל או ותק

:Route

GET /employees/age-or-seniority

מחזיר עובדים אשר עומדים בתנאי גיל חריג או ותק חריג.

5. מנהלים שאינם במחלקות מסוימות

:Route

GET /employees/managers/excluding-departments

מחזיר עובדים בתפקידי ניהול שאינם שייכים למחלקות שהוגדרו להחרגה.

6. עובדים לפי שם משפחה וגיל

:Route

GET /employees/by-lastname-and-age

מחזיר עובדים בהתאם לשם משפחה וגיל.

דגשים חשובים ל-`main.py`

- אין לכתוב לוגיקת שאילתות בקובץ זה
 - כל Endpoint:
 - קורא לפונקציה אחת בלבד מ-`dal.py`
 - מחזיר את התוצאה כפי שהתקבלה
 - אין שינוי במבנה הנתונים המוחזר
-

עבודה עם Git ו-Banches

Branches נדרשים (דרישת חובה)

על הריפוזיטורי להכיל לפחות את שלושת ה-Banches הבאים:

- **main** – מכיל את הקוד הסופי לבדיקה
אין לעבוד ישירות על Branch זה אלא למזג לתוכו רק קוד שעובד
- **dev** – Branch עבודה לפיתוח קוד האפליקציה
- **k8s** – Branch עבודה לקבצי YAML והגדרות פריסה

קיום שלושת ה-Banches האלו הוא דרישת סף.

צורת עבודה – חובה

- אסור לעבוד ישירות על **main**

- כל הפיתוח מתבצע על:

- **dev** (קוד אפליקציה)

- **k8s** (קבצי פריסה)

- מיזוג (**merge**) ל-**main** מתבצע עבור קוד שעובד ונבדק

שימו לב: רק התוכן שב-**main** ייבדק

צורת עבודה זו היא דרישת חובה, ואי-עמידה בה תגרור הפחתת ניקוד.

Branches נוספים – מומלץ

לצורך עבודה מסודרת ומקצועית, מומלץ (אך לא חובה) ליצור Branches נוספים, לדוגמה:

- **feature/dal**

- **feature/api**

- **feature/connection**

שימוש ב-Branches נוספים נחשב הרגל עבודה נכון, ומשפר את היכולת לנהל את הקוד ולאבחן באגים באפליקציה.

בדיקה וניקוד

- **main** – דרישת סף בלבד ומיקום הקוד שייבדק בפועל

- **dev** ו-**k8s** – קיומם מהווה חלק מן הציון הכולל - חוסר באחד מהם יגרור הפחתת ניקוד
-

פריסה ל-OpenShift

בשלב זה עליכם לפרוס את המערכת לסביבת **OpenShift** באמצעות קבצי **YAML** שתכתבו בעצמכם.

עליכם לממש את כל קבצי ה-YAML הנדרשים על מנת:

- להריץ את שרת האפליקציה
- להריץ מסד נתונים
- לאפשר תקשורת בין השרת למסד הנתונים
- לאפשר גישה לאפליקציה מתוך הדפדפן

דרישות פריסה

- יש להגדיר **Replica אחת בלבד**:
 - לשרת האפליקציה
 - למסד הנתונים
- אין צורך בהתאמות לביצועים או סקילינג
- הפריסה צריכה לאפשר הרצה תקינה של המערכת מקצה לקצה

טעינת נתונים

קוד לטעינת הנתונים למסד הנתונים יסופק לכם, למעט פרטי ההתחברות אותם תצטרכו לוודא ולשנות לפי הצורך (למשל באמצעות משתני סביבה, וחשיפת מסד הנתונים לגישה חיצונית לצורך הטעינה).

דגשים חשובים

- כל קבצי הפריסה יפותחו ב-`Branch k8s` (כאשר לבסוף ימוזגו ל-`main`)
 - אין לפרוס באמצעות הממשק הגרפי של OpenShift
 - הפריסה חייבת להתבצע באמצעות קבצי YAML
-

מרכיבי הניקוד:

- קבצי YAML עבור שרת האפליקציה
- קבצי YAML עבור מסד הנתונים MongoDB
- צילום מסך

צילומי מסך

בסיום הפריסה, עליכם להעלות **2 צילומי מסך**:

1. כל ה-Pods רצים

- צילום מסך המראה שכל ה-Pods בפרויקט במצב Running.
- יש לוודא שניתן לראות את שמות ה-Pods והסטטוס שלהם.

2. האפליקציה רצה בדפדפן

צילום מסך של האפליקציה פתוחה בדפדפן.

- יש להציג את שורת ה-URL המלאה
- יש לראות בבירור את הכתובת שנוצרה על-ידי OpenShift, כך שיהיה ברור כי ניגשנו ל-Route של האפליקציה (ולא localhost)