

מבחן MongoDB

פרויקט "מAGER העובדים"

Operation Silent Payroll

בסיוף שנת 2025 התקבלו במוסד התרבות מגורמי מודיעין ערביים על חברת טכנולוגיה בינלאומית בשם **Helios Dynamics Ltd**.

החברה פועלת כחברת הייטק לgitימית לכל דבר – פיתוח תוכנה, ייעוץ ו-Data Services – עם מאות עובדים, מחלקות מסודרות וশכורות תואמות שוק.

עם זאת, ניתוח ראשוני העלה חשד כי חלק מהעובדים בחברה עשויים להיות **שתולים איראניים**, הפעלים תחת CISI אזרחי ומנהליים קשורים לוגיסטיים וכספיים עבור רשתות טרור בינלאומיות.

מAGER העובדים של החברה, המכול נתונים על תפקידים, מחלקות, ותק ושכר, הועבר ליחידת הסיבר של המוסד לצורך תחקור ראשון.

המשך שלכם:

המשך שלכם היא לבנות מערכת חוקית שתנתנה לאגר נתונים ארגוני של עובדים, במטרה:

- לארח דפואים חריגם
- לזרות קבוצות עובדים חשודות
- ולהפיק תובנות ראשוניות מתוך נתונים שנראים לכורה שגרתיים

המערכת תבצע ניתוח על בסיס שאלות ותציג את התוצאות דרך מסך קיימ.

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

כל שאלתה שאתם ממשיכם מייצגת פולה חוקית לזייהו תבניות ארגוניות חריגות.

כלומר, כל תוצאה עשויה להיות קצרה.

טכנולוגיות ומימוש

המערכת תכלול את הרכיבים הבאים:

- שרת Backend מבווק **FastAPI**
- מסד נתונים **MongoDB** (הרצה באמצעות Container)
- סביבת הריצה ופריסיה:
 - (לסביבה ענן) **OpenShift**

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

דרישות המערכת

עליכם למשם מערכת צד-שרת אשר:

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

המערכת נדרשת:

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

כל פעולה במערכת מייצגת שלב בתהליכי 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-**כליז AI מכל סוג**
 - (כולל אך לא מוגבל ל-ChatGPT, Copilot, Claude, Gemini ובודומה)
 - קבלת עזרה ממAdam אחר
-

עזרים מותרים

מותר להשתמש אך ורק במקורות הבאים:

Phyton

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 (בדיקות 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)

Stack Overflow: •
[/https://stackoverflow.com](https://stackoverflow.com)

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 לנ נתונים
- **k8s/** – קבצי YAML לפרסה
- **Dockerfile** – בניית האפליקציה
- **requirements.txt** – תלויות הפרויקט
- **routes.py** – אופציונלי: קובץ נפרד להגדרת נקודות הקצה של השרת

קובץ `connection.py` – חיבור למסד הנתונים

בשלב זהה עיליכם למשח חיבור פעיל למסד הנתונים.
הקובץ `connection.py` אחראי על:

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

יצירת מסד הנתונים וה-Collection

יצירת מסד הנתונים וה-Collection היא **לא לבחירתכם**:

- ניתן ליצור אותם אוטומטית דרך הקוד
- ניתן להסתמך על יצרה אוטומטית של DB 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 **אוף הטעינה נתון לבחירתכם**.

מותר להשתמש בכל דרך נוחה:

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

dagshim chosibim

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

קובץ **dal** – פונקציות נדרשות

שימוש לב: את הלוגיקה של כל הפונקציות יש למש用人 במציאות שאילותות מול שירות MongoDB באמצעות חיבור פיתוני. אין למש用人 השאלות באמצעות לוגיקה פיתונית 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 במחלקה **Engineering**

כתבו שאילתת שמחזירה את פרטי העובדים הבאים:

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

עבור עובדים אשר:

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

get_employees_by_age_and_role

שאלה 2 – עובדים לפי גיל ותפקיד

כתבו שאילתת שמחזירה את כל פרטי העובדים אשר:

- גילם בין **30** ל-**45** (כולל את שני הקצוות)

- והתפקיד שלהם הוא **Specialist** או **Engineer**
-

`get_top_seniority_employees_excluding_hr`

שאלה 3 – עובדים עם ותק גבוה (ללא HR)

כתבו שאלתה שמחזירה את כל פרטי העובדים כאשר:

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

`get_employees_by_age_or_seniority`

שאלה 4 – עובדים לפי גיל או ותק

כתבו שאלתה 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 שאלתה 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 שאלתה שמחזירה את:

- name** •
- age** •
- job_role.department** •

עבור עובדים אשר:

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

קובץ `main.py` – REST API Endpoints

עליכם למשר שרת REST API אשר חושף את נקודות הヅה הבאות.
כל נקודהヅה מבצעת קרייה לפונקציה המתאימה בקובץ `dal.py` ומחזירה JSON תקין.

כל ה-Endpoints:

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

1. עובדים עם שכר גבוה במחלקה Engineering

:Route

`GET /employees/engineering/high-salary`

מחזיר עובדים ממחלקה Engineering עם שכר גבוה מוסף שנקבע.

2. עובדים לפי גיל ותפקיד

:Route

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

מחזיר עובדים בהתאם לטווח גיל ותפקיד עבודה מוגדרים.

3. עובדים עם ותיק גבוה (ללא HR)

:Route

GET /employees/top-seniority

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

4. עובדים לפי גיל או ותיק

:Route

GET /employees/age-or-seniority

מחזיר עובדים אשר עומדים בתנאי גיל חריג או ותיק חריג.

5. מנהליים שאינם במחלקות מסוימות

:Route

GET /employees/managers/excluding-departments

מחזיר עובדים בתפקיד ניהול שאינם שייכים למחלקות שהוגדרו להחלה.

6. עובדים לפי שם משפחה וגיל

:Route

GET /employees/by-lastname-and-age

מחזיר עובדים בהתאם לשם המשפחה וגיל.

dagשים חשובים לـ `main.py`

- אין לכתוב לוגיקת שאילתות בקובץ זה
 - כל Endpoint:
 - קורא לפונקציה אחת בלבד מ-`myдал`
 - מוחזיר את התוצאה כפי שהתקבלה
 - אין شيء במבנה הנתונים המוחזר
-

עבודה עם Git ו-`Branches`

`Branches` נדרש (דרישת חובה)

על הריפורזיטורי להכיל לפחות את שלושת ה-`Branches` הבאים:

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

קיים שלושת ה-`Branches` האלוי הוא **דרישת סף**.

צורת עבודה – חובה

- אסור לעבוד ישירות על `main`
- כל הפיתוח מתבצע על:
 - קוד אפליקצייה (`dev`)
 - קבצי פריסה (`k8s`)
- מיזוג (`merge`) ל-`main` מתבצע עבור קוד שעובד ונבדק

שימוש לב: רק התוכן שב-`main` יבדק

צורת עבודה זו היא דרישת חובה, ואינטימית בה תגרור הפקחת ניוק.

Branches – 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 יכולת הנהל את הקוד ולאבחן באגים באפליקציה.

בדיקות וניקוד

- דרישת סף בלבד ומיקום הקוד שיבדק בפועל – `main`
 - קיומם מהוות חלק מן הציון הכללי – חוסר באחד מהם יגרור הפקחת ניוק `dev` ו-`k8s`
-

פרישה ל-OpenShift

בשלב זה עליכם לפרש את המערכת לSUBSTITUTE OpenShift באמצעות קבצי YAML שתכתבו בעצמכם.
עליכם למש 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 פרישה

- יש להגדיר **אחד בלבד**:
 - לשרת האפליקציה
 - למסד הנתונים
 - אין צורך בהתקומות לביצועים או סקיליניג
 - הפרישה צריכה לאפשר הריצה תקינה של המערכת מוקצה לקצה
-

טעינת נתונים

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

דגשים חשובים

- כל קבצי הפרישה יפותחו ב-`s8` Branch (כאשר לבסוף ימואגו ל-`main`)
 - אין לפרסום באמצעות הממשק הגרפי של OpenShift
 - הפרישה חייבת להתבצע באמצעות קבצי YAML
-

מרכיבי הניקוד:

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

צילומי מסך

בסיום הפרישה, 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 והսטטוס שלהם.

2. האפליקציה רצתה בדף

צילום מסך של האפליקציה פתוחה בדף.

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