מבוא למחשוב ענן - סמסטר אביב התשפ"ה - תרגיל בית 3

**חלק ראשון**

|  |  |  |
| --- | --- | --- |
| שם חבר הצוות | משימות שהוקצו | משימות שהושלמו |
| לידור בן חמו | בדיקות התחברות, KPI | KPI בדיקות התחברות |
| ארד הרוש | משימות במסך עובדים ומנהלים | מימוש משימות במסכים |
| תומר רול - מהנדס מערכת | מערכת התחברות והרשמה | מימוש מערכת התחברות |
| טום ביטון | מערכת התחברות והרשמה | מימוש התחברות  והרשמה |
| יהלי רפפורט | צאט בוט | מימוש צאט בוט וקובץ וורד |

**Acceptance test – בדיקת קבלה**

|  |  |  |
| --- | --- | --- |
| **מספר בדיקה** | **תיאור בדיקה** | **תוצאה** |
| 1 | התחברות עם שם משתמש וסיסמה תקינים | הצלחה |
| 2 | התחברות עם סיסמה שגויה | הצלחה (שגיאה מוצגת כראוי) |
| 3 | רישום משתמש חדש עם כל השדות הנדרשים | הצלחה |
| 4 | רישום עם שם משתמש שכבר קיים | הצלחה (שגיאה מוצגת כראוי) |
| 5 | שליחת משימה לעובד דרך מסך מנהלים | הצלחה |
| 6 | סימון משימה שהושלמה ע"י עובד | הצלחה |
| 7 | הצגת דוח משימות במסך עובד | הצלחה |
| 8 | שליחת הודעה בצ'אט בוט וקבלת תגובה | הצלחה |
| 9 | לאחר לחיצה על כפתור הצג גרפים. הגרפים יעלו תוך פחות משלוש שניות. | הצלחה (בממוצע הגרפים עולים עד שלוש שניות) |
| 10 | חיפוש שתי מילים שאחת מהן לא מופיע במאגר יתבצע חיפוש עבור המילה המופיע ועבור המילה שלא נמצאת במאגר נתונים מופיע הודעת מתאימה. | הצלחה |

**חלק שני**

**בפרויקט השתמשנו ב microservices הבאים:  
1.שירות Search-Index**

מה הוא עושה: בונה אינדקס הפוך (‎term → docs‎), ממיר אותו ל-DataFrame ומעלה לקולקציה ייעודית ב-Firebase.

למה זה מיקרו-סרוויס: תחום אחריות צר וברור לניהול חיפוש, עם API ודאטה-סטור משלו, ולכן ניתן להריץ אותו כקונטיינר עצמאי.

**2.שירות Search-Analytics**

מה הוא עושה: מגדיל אסינכרונית את מוני החיפושים לכל מונח ושומר את הסטטיסטיקה בבסיס נתונים ייעודי.

למה זה מיקרו-סרוויס: נטול-מצב למעט כתיבה לדאטה-סטור שלו, פועל כ-API קטן שקל לשכפל לפי עומס.

**3.שירות Sensor-Data**

מה הוא עושה: שולף קובצי ‎JSON‎ של נתוני חיישנים לפי קטגוריה, מפרש אותם ל-DataFrame מסודר בזמן ומחזיר ללקוחות.

למה זה מיקרו-סרוויס: מחזיק מקור נתונים ייחודי ופורמט קבוע, עם גבול אחריות ברור וממשק אחיד.

**4.שירות User**

מה הוא עושה: מבצע פעולות ‎CRUD‎ מלאות למשתמשים (יצירה, שליפה, עדכון, מחיקה) בקולקציית ‎/users‎.

למה זה מיקרו-סרוויס: מנהל מודל זהויות עצמאי ו-API ברור, כך ששינויים בו אינם משפיעים ישירות על שירותים אחרים.

**5.שירות Task & Scoring**

מה הוא עושה: מאחסן משימות, מחשב ניקוד והישגים ומעדכן אותם בקולקציית ‎/tasks‎.

למה זה מיקרו-סרוויס: מטפל בדומיין גיימיפיקציה מובהק עם בסיס נתונים משלו ו-API נפרד.

**6.שירות Chat-Bot**

מה הוא עושה: עוטף את הקריאות ל-Gemini (Google Generative AI), שומר היסטוריית שיחה ומחזיר תגובות.

למה זה מיקרו-סרוויס: אינטגרציה חיצונית מבודדת; החלפת ספק או מפתח אינה דורשת שינוי בשאר המערכת.

**בפרויקט שלנו יש כמה KPI שיכולים לעזור למדוד את ההצלחה והשימושיות של המערכת:**

**1. חוויית משתמש (User Experience)** בפרויקט שלנו קיימת מדידה של זמן תגובה (Latency) של הבוט הג’מיני – מדד זה נאסף כחלק מהמיקרו-שירות שמטפל בתקשורת עם הבוט (Gemini Bot Integration). זמן תגובה ארוך עלול לפגוע בשטף השיחה ולתסכל את המשתמש. בנוסף, לוח המובילים (Leaderboard) מציג ניקוד ממוצע של המשתמשים, שמעיד על תחושת הצלחה, שביעות רצון ומעורבות. ככל שהניקוד גבוה יותר, כך ניתן להניח שהמשתמשים מוצאים ערך ומשתמשים בפלטפורמה באופן חיובי.

**2.גמישות ויכולת התרחבות (Scalability & Elasticity)** הפרויקט כולל טעינה שוטפת של נתוני חיישנים ממקורות חיצוניים (במיקרו-שירות Load Sensor Data), ונמדדת בו תדירות העדכון – כלומר, כמה רשומות מתקבלות לדקה. קצב עדכון גבוה בצורה יציבה מעיד על כך שהמערכת יודעת להתמודד עם זרימה מתמדת של נתונים מבלי לקרוס. זהו מדד מרכזי שמוכיח שהמערכת מסוגלת לגדול לפי הצורך ולהגיב לשינויים בעומס.

**3.זמינות ואמינות (Availability & Reliability)** במהלך טעינת נתונים מה־Firebase או מה־API של Gemini, יש התמודדות עם מצבים של שגיאה – לדוגמה, טיפול בתשובה ריקה או fallback להצגת ממשק ריק כאשר הבקשה נכשלת. התנהגות זו שומרת על המשכיות החוויה ומונעת קריסה של המערכת. בכך, המערכת שואפת להיות זמינה ואמינה גם כאשר יש בעיות זמניות ברשת או בשרת.

**4.עלות וניצול משאבים (Cost & Resource Utilization)** הפרויקט עושה שימוש מדוד ויעיל בקריאות API, במיוחד בטעינת אינדקס או בתהליך עיבוד השפה (NLP). לדוגמה, רק בעת הצורך נטענים נתונים חדשים או נשמרים שינויים. בנוסף, בניית האינדקס מתבצעת מקומית ורק לאחר מכן מועלה ל־Firebase – מה שמקטין את מספר הקריאות לרשת ומשפר את ניצול המשאבים. זהו היבט חשוב בכל פרויקט ענן, כדי לשמור על עלויות נמוכות ויעילות גבוהה.

**5.אבטחה ועמידה בדרישות רגולציה (Security & Compliance)** קיימת הבחנה בין סוגי משתמשים (עובד / מנהל), ושמירה על session בהתאם (באמצעות משתנים current\_user\_type, current\_username).

**שקיפות אלגוריתמית**בפרויקט שלנו הבהרנו למשתמשים כיצד האלגוריתמים פועלים ואילו נתונים נאספים בכל חלק של המערכת. במנוע החיפוש, המשתמש רואה עבור כל מונח באילו מסמכים הוא מופיע, כמה פעמים, והאם הוא מופיע בכותרת – כך שהחישוב שמאחורי תוצאות החיפוש ברור לגמרי. בנוסף, מצוין שהמערכת שומרת את מספר הפעמים שחיפשו כל מונח (searchCount) רק לצורך הצגת מונחים פופולריים, בלי לאסוף מידע אישי.  
 במסך הגרפים, המשתמש שולט בקצב הדגימה (כל שעה, כל 10 דקות וכו') ורואה כיתוב ברור מתחת לכל גרף שמסביר מה נמדד ומה המשמעות – מה שתורם להבנה של התמונה הכללית.  
 גם בצ'אטבוט (Gemini), המשתמש מקליד שאלה ורואה את התשובה המלאה, יחד עם תזכורת שהשאלה נשלחת למודל חיצוני (Gemini של Google) – כלומר יש שקיפות גם לגבי מקור המידע.  
גם בטבלת הניקוד של העובדים (Leaderboard), מוצגת לכל עובד שקיפות מלאה לגבי המשימות שביצע וכמות הנקודות שצבר.

כל אלה יחד עוזרים למשתמש להבין איך המערכת פועלת, מה נאסף, ומה מוצג – בצורה ברורה ושקופה.

**ניתוח מידע עתק**

בפרויקט שלנו יישמנו את מודל MapReduce לצורך עיבוד וניתוח של מידע עתק .

במסגרת הפרויקט, לכל מילה קיימת רשומה באינדקס שבה מאוחסן מידע אודות המסמכים שבהם היא מופיעה, מספר הפעמים שבהן היא מופיעה בכל מסמך ומספר המסמכים הכולל שבהם היא נמצאת.

יישמנו את שלושת שלבי MapReduce באופן הבא:

•    שלב המיפוי (Map): כל מסמך עובר סריקה, וכל מילה מנותחת ונשמרת עם מפתח המזהה את המסמך ואת מספר ההופעות שלה בו.

•    שלב הקיבוץ (Shuffle): לאחר איסוף הנתונים, מתבצע קיבוץ של כלל המופעים של כל מילה מכלל המסמכים, כך שניתן לרכז את כל הנתונים הרלוונטיים לאותה מילה.

•    שלב החישוב (Reduce): עבור כל מילה מתבצעת סכימה של כלל ההופעות שלה, והמידע משמש להצגת נתונים למשתמש, כולל דירוג תוצאות והצגה של מספר ההופעות הכולל של המילה בכל דפי האינטרנט יחד.

באמצעות שימוש במודל MapReduce הצלחנו להתמודד בצורה יעילה עם עיבוד טקסטים בהיקף נרחב, תוך שמירה על ביצועים גבוהים ויכולת גידול עתידית של המערכת. המודל מאפשר לנו לנתח מידע בצורה מבוזרת ואפקטיבית.

**אתגרים שעלו במהלך העבודה**

במהלך הפרויקט נתקלנו במספר אתגרים, במיוחד כי זו הייתה הפעם הראשונה שעבדנו עם Google Colab. אחד הקשיים המרכזיים היה עבודה משותפת על אותו קוד – כשעבדנו במקביל, לפעמים חלקים מהקוד נמחקו או שונו בטעות, מה שדרש מאיתנו לתאם ולנהל גיבויים בצורה מסודרת.

בנוסף, לקח לנו זמן להבין איך בדיוק מתבצע החיבור לסנסורים ואיך הנתונים מה־Firebase משתלבים עם הגרפים, מה שיצר בלבול בשלבים הראשונים של הפרויקט.

גם ברמה הוויזואלית היה אתגר – רצינו שהמערכת תיראה טוב ונעבוד עם עיצוב נעים, אבל בקולאב זה היה פחות אינטואיטיבי לעומת עבודה על ממשק רגיל עם קוד HTML/CSS. לכן לקח זמן עד שהגענו לעיצוב שאנחנו מרוצים ממנו.

למרות האתגרים, הצלחנו להתגבר עליהם תוך כדי למידה והתייעצות, והפרויקט הפך למערכת מלאה שעובדת בצורה יציבה וידידותית למשתמש.

**חלק שלישי : סגירת הפרויקט**

**תיק מתכנת**

1. **קבצים מרכזיים:**

מחברת collab של הקוד והרצה - [HM3 - CLOUD.ipynb](https://colab.research.google.com/drive/1EoZVMQki2m6-gSGLD1TI3ixmJTnjKcDS?authuser=1" \l "scrollTo=GDYS0mYkmz1Y)

Notebook זה משמש אב־טיפוס למערך מיקרו‑שירותים במערכת ענן המספקת חיפוש, אנליטיקה, ניהול נתוני חיישנים, ניהול משתמשים, גיימיפיקציה ובוט AI. מטרת תיק המתכנת היא למסמך את מבנה הקוד, התלויות, נקודות ההרחבה וה‑API הפנימיים כך שמפתחים חדשים יוכלו להתמצא ולהמשיך פיתוח ותחזוקה.

**2. מבט‑על ארכיטקטוני**

ליבה: Jupyter Notebook המקפל בתוכו תאי קוד שכל אחד מהם מייצג שירות או רכיב Orchestration.

Data Layer: Firebase Firestore/Storage משמש כבסיס נתונים מרכזי לכל השירותים. כל שירות ניגש לקולקציה ייעודית.

Integration Layer: ספריית `google-generativeai` משלבת את Gemini לצורכי צ‘אט בוט.

Deployment Target: כל שירות ניתן לאריזה כ‑Container עצמאי (Docker או Cloud Functions) ולחיבור דרך REST או Pub/Sub.

**3. תלויות עיקריות**

\* Python 3.10+

\* `firebase-admin >= 6.4.0`

\* `pandas`, `numpy`, `google-cloud-storage`

\* `google-generativeai` (ל‑Gemini)

\* `ipython\_widgets` (ל‑UI מקומי)

\*הערה: מומלץ להשתמש ב‑`pipenv` או `poetry` לנעילת גרסאות.

**4. מבנה נתונים – Firestore**

**1. mqtt\_index**

מאגר אינדוקס למונחים מתוך טקסטים או מסמכים, ככל הנראה לשימוש פנימי בחיפוש.

term – המונח עצמו (למשל "able").

DocIDs – אובייקט המכיל מידע על הופעת המונח:

TotalCount – מספר כולל של מסמכים שבהם המונח מופיע.

docCount – כמה פעמים המונח מופיע במסמכים מסוימים.

perDocInfo – מידע נוסף לפי מסמך:

searchCount – מספר הפעמים שהמשתמשים חיפשו את המונח הזה.

**2.indoor**

מידע מחיישן פנימי בטופיק MQTT של החדר D106.

Distance – מרחק שנמדד (יחידות לא ידועות – ייתכן בס"מ או מ"מ).

timestamp – תאריך ושעה של המדידה.

topic – מחרוזת שמייצגת את הטופיק ב־MQTT שממנו התקבלה המדידה (במקרה הזה: "indoor").

**3. outdoor**

(בהנחה שמדובר במבנה דומה ל־indoor) – מדידות חוץ סביבתיות.

DLight – עוצמת אור (Daylight), נמדדת על ידי חיישן.

Humidity – אחוזי לחות באוויר.

Pressure – לחץ אטמוספרי.

Temperature – טמפרטורה חיצונית.

timestamp – תאריך ושעה של המדידה.

topic – הטופיק של MQTT שממנו הגיעו הנתונים ("outdoor").

**4. scores**

ניקוד של משתמשים במערכת.

מפתח (למשל: tomer1) – שם המשתמש.

ערך – מספר הנקודות שצבר המשתמש.

**5. tasks**

מערכת ניהול משימות לפי משתמשים.

מפתח ראשי – שם המשתמש (למשל lidor).

מזהה ייחודי (UUID) – מזהה של כל משימה.

שדות של כל משימה:

task – תיאור המשימה לביצוע.

parameter – על איזה פרמטר מתמקדת המשימה (למשל Humidity).

points – מספר נקודות שהמשימה שווה אם תושלם.

completed – האם המשימה הושלמה (true או false).

note – הערות נוספות על המשימה (יכול להיות ריק).

**5. הוראות הפעלה מקומית**

1. הורד את קובצי הפרויקט (‎HM3\_CLOUD.ipynb‎) לסביבת העבודה המקומית.

2. צור דאטה בייס בעזרת firebase ותייבא את הurl. לאחר מכן תבצע השמה לתוך משתנה base\_url של הurl שלך.

3. מאחר שאין קובץ `requirements.txt`, התקן את התלויות ישירות:

```bash

# בקולאב אין צורך בהתקנה אם הספריות כבר זמינות. במידת הצורך, ניתן להריץ:

!pip install firebase-admin pandas numpy google-cloud-storage google-generativeai ipywidgets

4. הרץ את microservice-build index ותוודא שנוצר האינדקס.

5. הרץ את הקוד.

**6.פונקציות מרכזיות:**

**ask\_gemini\_bot(prompt)** –

שואל את מודל Gemini שאלה ומחזיר את התשובה.

**fetch\_full\_index()** –

טוען את כל האינדקס מ־Firebase ומחזיר אותו כ־DataFrame.

**increment\_search\_count(term)** –

 מעלה את מונה החיפושים של מונח מסוים ב־Firebase.

**show\_search\_screen()** –

 מציג ממשק לחיפוש מונחים באתר כולל תמיכה ב־AND/OR, התאמה מדויקת, ושילוב עם Gemini.

**show\_top\_search\_terms\_screen()** –

 מציג את עשרת המונחים הכי מחופשים בטבלה מעוצבת.

**show\_stats\_screen()** –

 מציג גרפים של נתוני חיישנים (טמפרטורה, לחות, לחץ) מה־Firebase.

**fetch\_data(category)** –

 שולף ומעבד נתוני קטגוריה (indoor/outdoor) מה־Firebase ל־DataFrame.

**show\_optimization\_dashboard()** –

 מציג לוח משימות לעובדים וממשק ניהול משימות למנהלים.

**show\_add\_task\_ui()** –

 מציג טופס להוספת משימה חדשה לעובד.

**show\_admin\_dashboard()** –

 מציג דשבורד מותאם לפי סוג המשתמש (מנהל/עובד) עם התראות, כפתורי ניווט, ומשימות יומיות.

**show\_Manage\_Employees\_screen()** –

 מאפשר הרשמה של עובדים חדשים, הצגת רשימה, ומחיקה.

**show\_employee\_tasks()** –

 מציג לעובד את כל המשימות האישיות שלו עם אפשרות לסימון ביצוע.

**show\_login\_screen()** –

 מציג טופס התחברות למערכת.

**7. קטעי קוד ותבניות עיצוב מעניינים**

במהלך העבודה על הפרויקט שילבנו כמה תבניות עיצוב ופתרונות קוד מעניינים שמדגימים תכנון מסודר וחשיבה מעשית.

אחד המרכיבים המרכזיים היה מנוע אינדוקס שפיתחנו בעצמנו. הוא אחראי לסרוק את דפי האתר mqtt.org, לעבד את הטקסטים באנגלית, לסנן מילים שכיחות, לזהות את סוגי המילים (למשל פועל או שם עצם), ולהחזיר עבור כל מונח באילו דפים הוא מופיע וכמה פעמים. זה מאפשר חיפוש יעיל ומדויק – עם שקיפות מלאה למשתמש.

בנוסף, השתמשנו ב־Threading כדי להריץ פעולות ברקע בלי להפריע לשימוש במערכת. למשל, כשמשתמש מחפש מונח, המערכת מעדכנת את מספר הפעמים שחיפשו אותו – אבל עושה את זה ברקע כדי לא להאט את החיפוש.

בצד של הממשק, השתמשנו בספריית ipywidgets יחד עם HTML מותאם כדי לבנות טפסים, טבלאות, וגרפים – בצורה אינטראקטיבית ונוחה, גם בלי להשתמש בכלים כבדים יותר כמו React או Flask. זה הפך את הממשק לדי מרשים יחסית לסביבת Jupyter.

אחד הדברים היותר מיוחדים שהוספנו היה חיבור ל־Gemini API של Google. כך המשתמש יכול לשאול שאלות חופשיות על MQTT ולקבל תשובות חכמות מתוך ממשק החיפוש עצמו. השילוב הזה דרש מאיתנו להבין גם איך עובדים עם API חיצוני וגם איך מציגים את זה בצורה חלקה בממשק עצמו.

**תיק משתמש**

**הסבר כללי על המערכת**

המערכת מהווה פתרון כולל לצפייה וניתוח נתוני חיישנים המגיעים בזמן אמת לצד חיפוש מונחים מתוך אתר mqtt. היא מאפשרת למשתמשים לצפות בנתוני חיישנים כמו טמפרטורה ולחות, לחפש מונחים מתוך אינדקס שנבנה אוטומטית מתוך האתר, לנהל משימות לעובדים, ולבצע פעולות ניהול כמו הוספה ומחיקה של משתמשים. המערכת בנויה על ממשק אינטראקטיבי ב־Jupyter Notebook תוך שימוש ב־ipywidgets ו־Firebase לניהול נתונים בענן. בנוסף, משולב במערכת צ'אטבוט חכם מבוסס Gemini API שמאפשר שאילת שאלות על עולם ה־MQTT.

**פירוט מסכים**

**מסך התחברות:** הכניסה למערכת מתבצעת דרך מסך התחברות הכולל שדות לשם משתמש וסיסמה. לאחר ההתחברות, המשתמש מועבר למסך הבית הרלוונטי לו – דשבורד לעובד או דשבורד למנהל. במקרה של פרטים שגויים, תוצג הודעת שגיאה מתאימה.

**דשבורד (ראשי):** למשתמשים מוצג דשבורד שונה בהתאם לסוגם:  
 עובדים רואים את המשימות האישיות שלהם ואת התראות הסביבה בזמן אמת.  
 מנהלים רואים בנוסף שגיאות אחרונות מהשבוע האחרון, יכולים לנהל עובדים, להוסיף משימות ולנתח סטטיסטיקות של הסביבה.

**מסך חיפוש מונחים:** מאפשר חיפוש חכם של מונחים שנמצאו בדפי האתר של mqtt.org. ניתן לבצע חיפוש מדויק או חיפוש חלקי, ולחפש מספר מונחים בו־זמנית. תוצאות החיפוש מציגות קישורים ישירים לדפים שבהם הופיע המונח, האם המונח נמצא בכותרת, וכמה פעמים הוא הופיע. בנוסף, במסך זה משולב צ'אטבוט חכם (Gemini) שמספק תשובות לשאלות כלליות.

**מסך מונחים פופולריים:** מציג את עשרת המונחים הכי מחופשים על פי מספר הפעמים שחיפשו אותם בפועל. כל שורה מציגה את שם המונח וכמות החיפושים שנעשו לו.

**מסך סטטיסטיקות חיישנים:** מציג גרפים אינטראקטיביים עבור מדדים סביבתיים כגון טמפרטורה, לחות ולחץ ברומטרי, הן בסביבה פנימית והן חיצונית. ניתן לשנות את קצב הדגימה של הנתונים (למשל כל שעה או כל 10 דקות). הגרפים מלווים בהסבר טקסטואלי לכל גרף.

**מסך משימות עובדים:** מציג לכל עובד את רשימת המשימות האישית שלו עם אפשרות לסמן אילו משימות הושלמו. עבור מנהל, ניתן להוסיף ולמחוק משימות לכל עובד דרך טופס מסודר. בנוסף, מוצגת טבלת דירוג (Leaderboard) של העובדים לפי מספר הנקודות שצברו ממשימות שבוצעו.

**מסך ניהול עובדים (למנהל בלבד):** מאפשר למנהל המערכת להוסיף עובדים חדשים עם שם משתמש וסיסמה, למחוק עובדים קיימים, ולצפות ברשימת כל העובדים הרשומים במערכת.

**מעברים בין מסכים**

המעברים בין המסכים מתבצעים באמצעות כפתורים ברורים במסך. למשל, מהדשבורד ניתן לעבור למסך החיפוש, הגרפים או המשימות בלחיצה אחת. אין צורך ברענון של הדף – כל המעברים הם דינמיים ומידיים.

**טעויות אפשריות והסבר**

* **שגיאת התחברות (שם משתמש/סיסמה שגויים):** תופיע הודעה מתאימה עם בקשה לבדוק שוב את הפרטים.
* **לא נמצאו תוצאות בחיפוש:** ייתכן שהמונח לא מופיע באינדקס – מומלץ לבדוק כתיב.
* **כשל בקבלת נתונים מ־Firebase:** יש לבדוק את חיבור האינטרנט או לנסות שוב לאחר מספר שניות.
* **שגיאה בצ’אט עם Gemini:** לרוב נובעת משאלה כללית מדי או עומס – ניתן לנסח מחדש ולשלוח שוב.

**שקיפות אלגוריתמית**

המערכת תוכננה כך שתהיה שקופה וברורה למשתמשים, הן מבחינת אופן פעולת האלגוריתמים והן מבחינת הנתונים שנאספים ומוצגים.

במנוע החיפוש, מוצגות למשתמש תוצאות הכוללות את רשימת המסמכים שבהם נמצא המונח, מספר ההופעות שלו בכל מסמך, והאם הופיע גם בכותרת הדף. מידע זה מאפשר להבין כיצד מתקבלות תוצאות החיפוש ואילו קריטריונים משמשים להכללת מונח באינדקס.

המערכת עוקבת אחר מספר הפעמים שמונח מסוים חופש (searchCount), אך ורק לצורך הצגת רשימת מונחים פופולריים. אין כל איסוף או שמירה של נתונים אישיים, ולא מתבצע מעקב אחר זהות המשתמשים.

במסך הגרפים, המשתמש בוחר בעצמו את קצב הדגימה של הנתונים (למשל, כל שעה או כל 10 דקות), ובכל גרף מופיע הסבר טקסטואלי קצר שמתאר את הנתון המוצג ואת משמעותו. כך נשמרת בהירות בניתוח הנתונים הסביבתיים.

בצ'אטבוט המשולב במערכת (המבוסס על Google Gemini), המשתמשים מקבלים תשובה ישירה לשאלותיהם, תוך הבהרה שהשאלה נשלחת לעיבוד במודל חיצוני. באופן זה נשמרת מודעות למקור המידע.

בנוסף, בטבלת הניקוד של העובדים מוצגת לכל עובד שקיפות מלאה: ניתן לראות אילו משימות בוצעו, כמה נקודות נצברו, ומה המיקום בדירוג. הדבר מעודד תחרות הוגנת ומחזק את תחושת הבקרה האישית.השקיפות זו מאפשרת למשתמש להבין בכל שלב כיצד מתקבלות ההחלטות במערכת, אילו נתונים נאספים, וכיצד הם משמשים – וכל זאת תוך שמירה על פרטיות וקלות שימוש.

**קישור לסרטון שמתאר את השימוש במערכת:** [**cloud\_eligator.mp4**](https://drive.google.com/file/d/1lH7jzDMXtHyASahbvH3XtMPVIjSOY58Y/view?usp=sharing)