**מבוא למחשוב ענן**

**שמות חברי הצוות:**

ימית מורנו 207476102, עומר זילחה 211695440, קארין ארמה 209702299 ,נוי בן עזרא 211565130, בר אלרום 209121573, יובל כץ 207504952

**קישור למחברת קולאב:**

<https://colab.research.google.com/drive/1KbjCNq8rkStciOdIXJJJAICe-JvvITug?usp=sharing>

**קישור לגיט:**

<https://github.com/KarynA19/Cloud-Computing>

**מהות המוצר ומרכיביו   
מודולים ופיצ'רים מעניינים**

### **פיצ’ר: הצגת תחזית מזג אוויר לכרמיאל והקשר לחיישני Outdoor**

בפרויקט הוספנו פיצ’ר פונקציונלי המציג תחזית מזג אוויר עדכנית ליום הנוכחי ולשלושת הימים הקרובים עבור העיר כרמיאל — שם ממוקמת המעבדה בה מותקנים חיישני החוץ (Outdoor sensors).

#### **מטרת הפיצ’ר:**

#### מטרת התחזית היא לאפשר למהנדסים ולמנהלים לצפות בתנאי מזג האוויר הנוכחיים והצפויים, ולוודא שהנתונים המתקבלים מהחיישנים החיצוניים תואמים את התנאים בסביבה החיצונית בפועל. כך ניתן לגלות חריגות אפשריות בטמפרטורה או בלחות ולהבין אם הן נובעות מהחיישן עצמו או מהשפעות חיצוניות (כגון עומס חום או רוחות חריגות).

**microservices שנעשה בהם שימוש**

במהלך פיתוח המערכת יישמנו את עקרונות ה־ **Microservices** בכך שפיצלנו את הלוגיקה לשירותים מבודדים, כאשר כל שירות אחראי על פונקציונליות אחת בלבד. השירותים פועלים בתהליכים נפרדים ומתקשרים ביניהם.

**שירות 1: Sensor Writer Service - כתיבת נתוני חיישנים למסד הנתונים**

שירות עצמאי אשר אחראי על קליטת נתוני חיישנים מסוג indoor ו־outdoor משרת ה־MQTT ועיבודם לטובת עדכון שוטף במסד הנתונים Firebase .

**מאפיינים עיקריים:**

* פועל כתהליך נפרד שרץ ברציפות ללא תלות בשירותים אחרים במערכת.
* מקשיב לנושאים רלוונטיים במערכת ה־MQTT ומטפל בכל הודעה שנכנסת באופן מיידי.
* מבצע עדכון של ערכי החיישנים בפורמט מובנה לתוך Firebase .

**תועלת למערכת:**

* מבטיח שהנתונים המתקבלים מהחיישנים יהיו זמינים בזמן אמת לשירותים אחרים כגון הצגה גרפית, ניתוחים סטטיסטיים, או פעולות משתמש.
* מאפשר הפרדה בין איסוף נתונים לעיבוד והצגה, וכך תורם לארכיטקטורה מודולרית וניתנת להרחבה.
* מייעל את ביצועי המערכת על ידי כך שאינו חוסם או מעכב את פעולת הממשק או שירותים אחרים.

**שירות 2: Search Engine Service - בניית אינדקס תוכן מאתרים חיצוניים**

שירות עצמאי אשר אחראי על שליפת תוכן טקסטואלי מאתרי אינטרנט, ניתוח מילות מפתח, ובניית אינדקס חכם לצורכי חיפוש. התוצר נשמר במסד הנתונים Firebase ומאפשר שאילתות חיפוש על בסיס תוכן אמיתי שנמצא באתרי אינטרנט.

**מאפיינים עיקריים:**

* מבצע קריאה לאתרים חיצוניים, חילוץ טקסט וניתוח מילים באמצעות ביטויים רגולריים.
* מסנן מילים נפוצות (Stop Words), סופר הופעות, ומבנה אינדקס הכולל כתובת URL וכמות הופעות לכל מילה.
* שומר את האינדקס במסד הנתונים Firebase בפורמט מותאם לחיפוש.
* מאפשר שליפת תוצאות חיפוש מדויקות לפי מילות מפתח, כולל דירוג לפי כמות הופעות ודיוק.

**תועלת למערכת:**

* מאפשר שילוב מנוע חיפוש תוכן מותאם אישית המבוסס על מידע דינמי מהאינטרנט.
* תורם להפרדת תחומי אחריות - אינדוקס מתבצע כשירות עצמאי, ואינו תלוי ב UI או בנתוני חיישנים.
* ניתן להרחבה בקלות לשירותים מתקדמים כגון ניתוח סמנטי, זיהוי ישויות, או המלצות חכמות.

**KPI מרכזיים**

### אמינות ודיוק הנתונים (**Data Accuracy KPI**)

הסבר: כדי להבטיח שהמערכת תשקף את המצב האמיתי במעבדה – נדרש לבדוק עד כמה ערכי החיישנים המוצגים באפליקציה תואמים למדידות האמיתיות. דיוק גבוה מונע תגובה שגויה ומחזק את אמון המשתמשים במערכת.

● **מה נמדד:** רמת ההתאמה בין נתוני החיישנים באפליקציה לבין מדידות ידניות בפועל (למשל מד טמפרטורה פיזי במעבדה).

● **איך נבדק:** באמצעות השוואה של קריאות נבחרות מול מדידה ידנית, ובחינת האם הן נמצאות בטווח סטייה מותר ( ±0.5°C).

● **יעד (SLA):** לפחות **99%** מהקריאות יהיו מדויקות ויכנסו לטווח הסטייה שהוגדר מראש.

### זמינות המערכת (**System Availability KPI**)

**הסבר:** מערכת שאינה זמינה בזמן אמת אינה מאפשרת ניטור תקין של תנאים קריטיים – מה שעלול להוביל לנזק. זמינות גבוהה מעידה על המשכיות תפעולית ועל אמינות בשטח.

● **מה נמדד:** אחוז הזמן שבו המערכת פועלת כראוי ונגישה למשתמשים, לאורך זמן (למשל במהלך חודש).

● **איך נבדק:** ניטור רציף של זמני פעילות מול זמני השבתה, לדוגמה על ידי לוגים או כלי ניטור שרתים (Firebase או מקומי), והשוואה מול זמן כולל.

●  **יעד (SLA):** זמינות של לפחות 99% מהזמן החודשי – כלומר מקסימום 7 שעות השבתה בחודש.

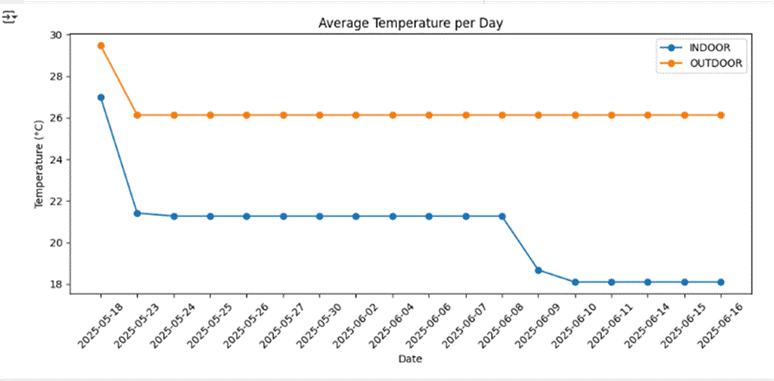
עדכניות הנתונים (**Data Freshness KPI**)

הסבר:נתונים שמתעדכנים באיחור פוגעים ביכולת לזהות חריגות בזמן אמת. KPI זה בודק את מהירות ההגעה של נתוני החיישנים לאפליקציה.

● **מה נמדד:** הזמן שעובר בין הרגע שהחיישן מייצר נתון חדש – לבין רגע ההצגה שלו באפליקציה בפועל.

● **איך נבדק:** על ידי חישוב הפרש בין Timestamp של שליחת הנתון ל־Firebase לבין זמן הרינדור/הצגה באתר, באמצעות לוגים או כלים מובנים בקוד.

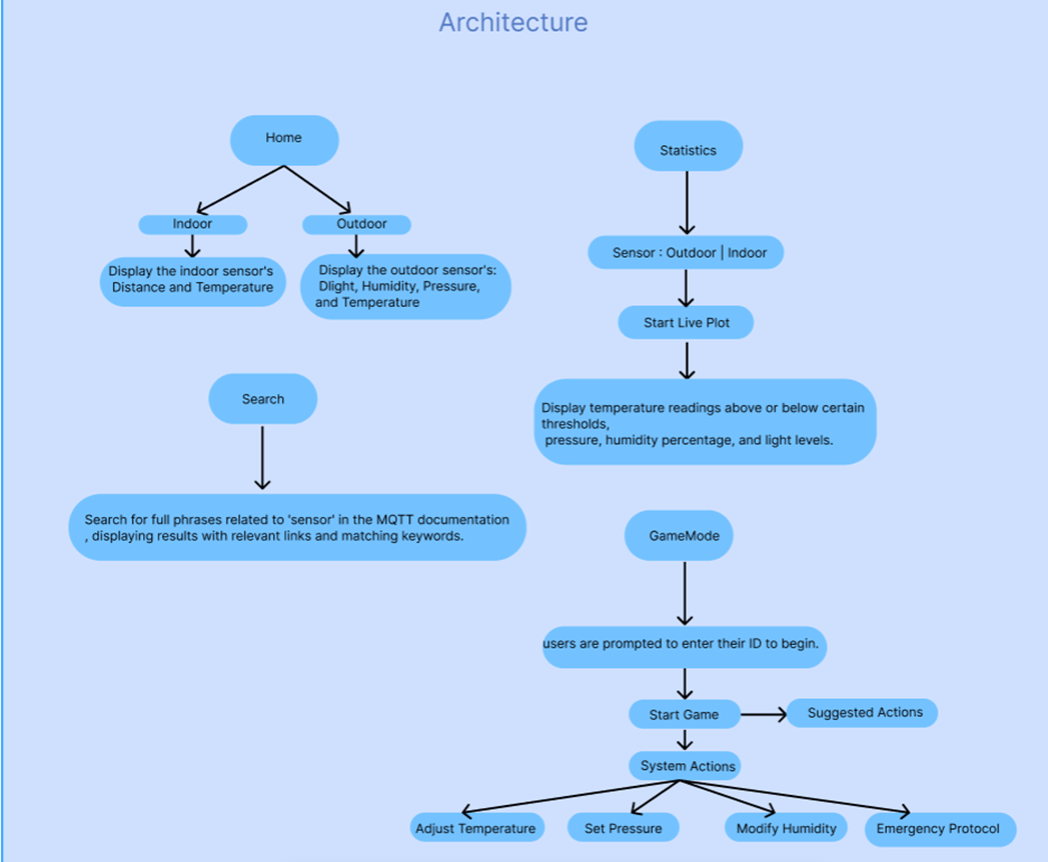
● **יעד (SLA):** לפחות **95%** מהנתונים יוצגו למשתמש תוך עד 5 שניות מרגע ההפקה.

**ניתוח נתוני עתק**

ביצענו ניתוח נתוני טמפרטורה מ־Indoor ו־Outdoor באמצעות Spark, שכלל MapReduce ו־groupBy לחישוב ממוצע יומי לכל מיקום. **ניתוח הגרף:** הגרף מציג ממוצע טמפרטורות יומי במשך כחודש.

* ב־18.5 נמדדו טמפרטורות גבוהות במיוחד (מעל 29° בחוץ וכמעט 27° בפנים) – כנראה אירוע חריג.
* מ־23.5 טמפרטורת החוץ מתייצבת, מה שמעיד על יציבות מזג האוויר.
* בפנים נרשמת ירידה חדה בטמפרטורה החל מ־8.6 – ייתכן בשל הפעלת מיזוג, שכן החוץ נותר יציב.
* הפער בין פנים לחוץ נותר קבוע עד 9.6 ואז גדל, מה שמרמז על שינוי אנושי בתנאי החדר.

**ארכיטקטורת המערכת**

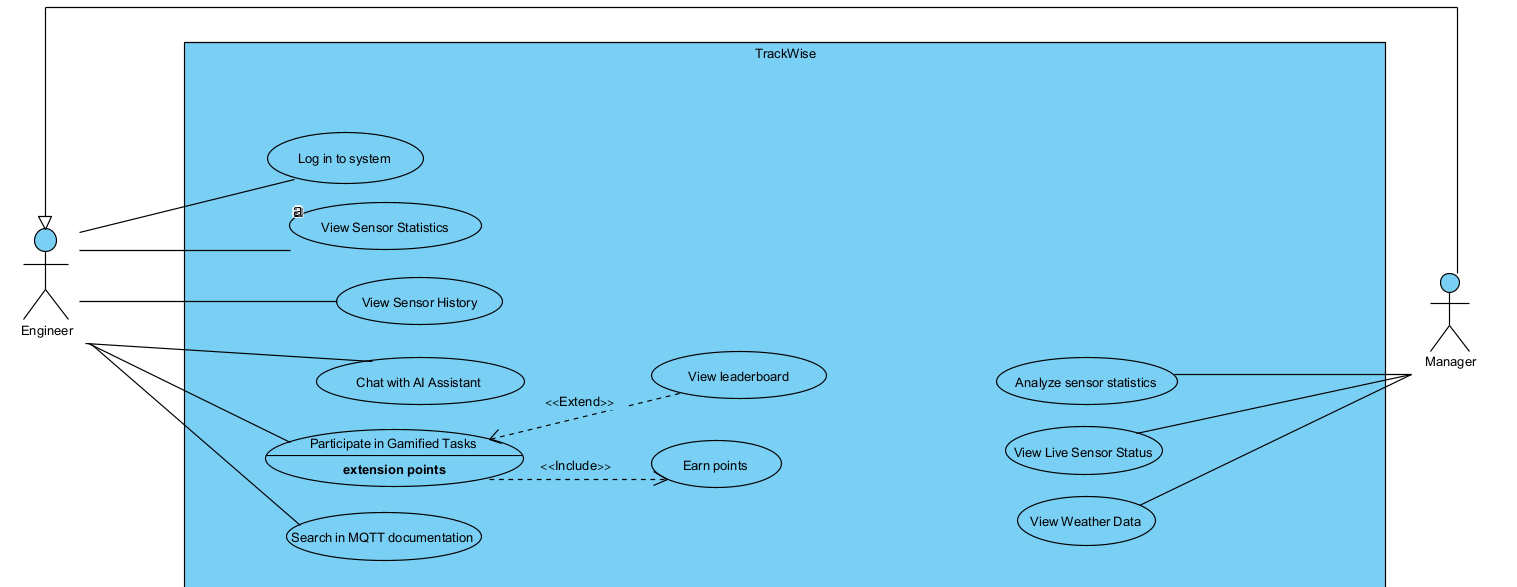


המערכת שלנו בנויה בארכיטקטורת **Layered Architecture**, המחלקת את המערכת לפי תחומי אחריות:

* **Presentation Layer** – מציגה נתונים למשתמש ומנהלת אינטראקציה (כגון גרפים, חיפוש).
* **Business Logic Layer** – מבצעת ניתוח חיישנים, מצב משחק ונתוני MQTT.
* **Data Access Layer** – מתקשרת עם Firestore ו־MQTT לשליפת נתונים.

הדיאגרמה מציגה זרימה בין השכבות, כאשר כל שכבה תלויה רק בזו שמתחתיה – דבר שמפשט תחזוקה ושדרוגים.  
 בנוסף, יש גם מימד טכני של **Client-Server**, שבו ה־Client שולח בקשות ל־Firebase, אך זהו היבט מימושי בתוך הארכיטקטורה הלוגית.

**Use Case**



**דרישות פונקציונליות**

* צפייה בנתוני חיישנים היסטוריים לפי סוג, תאריך ושעה, עם הצגה בטבלה.
* קליטת נתוני חיישנים פנימיים וחיצוניים בזמן אמת למסד Firebase.
* הצגת ערכי חיישנים עדכניים וזמן עדכון במסך המנהל.
* חיפוש טקסטואלי חופשי עם תוצאות לפי רלוונטיות.
* צ’אט עם מודל בינה מלאכותית (Gemini).
* תחזית מזג אוויר ל־4 ימים קדימה לעיר כרמיאל.
* יצירת גרפים אינטראקטיביים לפי טווח תאריכים, שעות ומיקום.

**דרישות לא פונקציונליות**

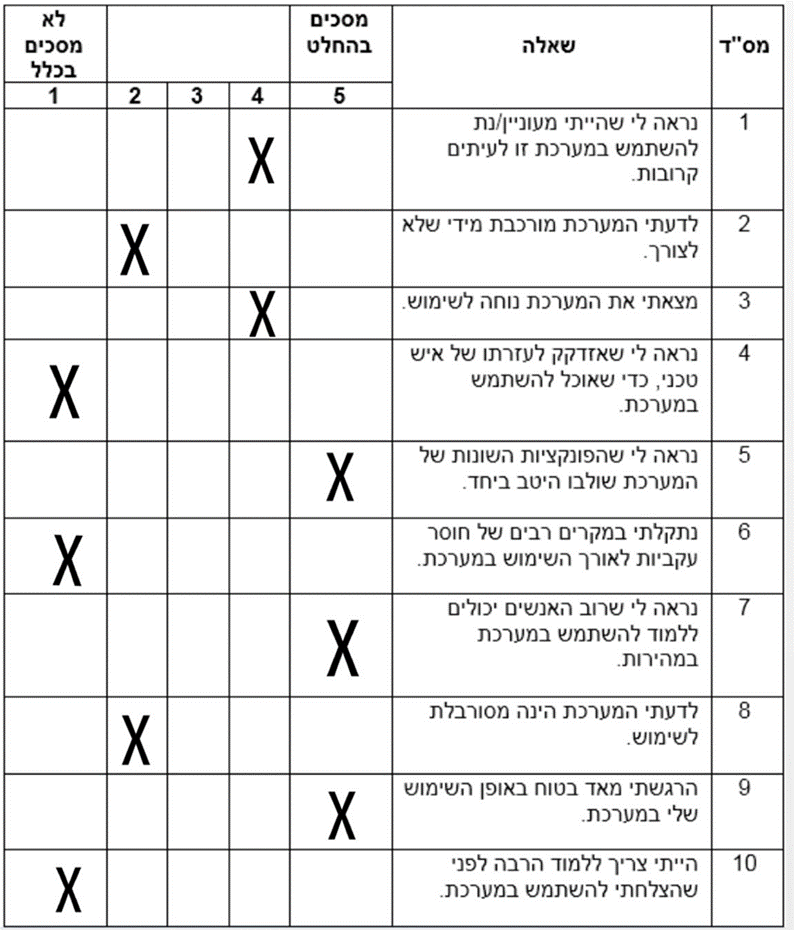
### 1. **Performance (ביצועים):** המערכת מציגה נתוני חיישנים בזמן אמת, עם עדכון כל 10 שניות. אתגרים: עיכובים בקבלת נתונים, ועומס על גרפים ודשבורד.

### 2. **Availability (זמינות):** המערכת חייבת לפעול ברציפות עבור משתמשים אונליין. אתגרים: תקלות ברשת, זמינות שרתי Firebase ו־MQTT.

### 3. **Scalability (יכולת התרחבות):** יש להיערך לתמיכה בחיישנים ומשתמשים נוספים. אתגרים: התאמת Firebase ו־Colab לגידול, ירידה בביצועים.

### 4. **Usability (שימושיות):** הממשק כולל תפריטים, גיימיפיקציה וצ'אט AI, ודורש שימוש נוח. אתגרים: התאמה לסוגי משתמשים שונים ופשטות בשדות ובהודעות שגיאה.

**ביקורת עמיתים אשר ניתנה במהלך הסטודיו (שבוע 9) , וכיצד התמודדתם איתה. נא לחשב גם את ציון SUS ולהתייחס אליו**.

מסמך **SUS** של המערכת 

ציון **SUS** מקומות זוגים: (4-1)+(4-1)+(5-1)+(5-1)+(5-1)=18   
מקומות אי זוגיים: (5-2)+(5-1)+(5-1)+(5-2)+(5-1) =18  
חישוב סופי: (18+18)\*2.5=90  
  
ציון ה־SUS שקיבלנו לא הפתיע אותנו, לאור חוסר הבהירות בתחילת הדרך לגבי הכלים והטכנולוגיות הדרושים. לאחר ייעוץ עם המרצה נטלי הבנו שהמערכת שבנינו לא התאימה לדרישות, והתחלנו מחדש דבר שפגע בלוחות הזמנים. למרות זאת, הגענו לסדנה עם מערכת שאינה מושלמת, אך מייצגת את המיטב שיכולנו להפיק בזמן הנתון.

**שקיפות אלגוריתמית (Algorithmic Transparency)**

במהלך הפיתוח שמנו דגש על **שקיפות** כדי להבטיח שהמשתמש יבין את פעולת המערכת, הנתונים הנאספים, והשפעת האלגוריתמים:

1. **נתוני חיישנים בזמן אמת:** הנתונים (טמפרטורה, לחות, לחץ אור) מוצגים בבירור לפי אזור, עם חותמת זמן, כך שהמשתמש רואה מה נאסף ומתי.
2. **הסבר ודירוג תוצאות חיפוש:** תוצאות כוללות מידע על כמות הופעות ודירוג לפי התאמה, כדי להבהיר למה תוצאה מופיעה בראש.
3. **שקיפות בניקוד (Game Mode):** לכל פעולה מוצגים מספר הנקודות והציון המצטבר, כך שהשפעת כל פעולה ברורה מיידית.
4. **הסברים טקסטואליים לצד נתונים:** המערכת מספקת פירושים לטווחים תקינים, חריגות וסטטיסטיקות – מה שמחזק את ההבנה והשליטה של המשתמש.

**אתגרים**

1. **שילוב טכנולוגיות רבות (MQTT, Firebase, NLTK, BeautifulSoup וכו'):** הפרויקט כלל עבודה עם מגוון רחב של ספריות וטכנולוגיות, שכל אחת מהן דורשת קונפיגורציה שונה. לדוגמה, חיבור ל-MQTT ו-Firebase דרש קריאה לתיעוד, טיפול בהרשאות, ותחזוקה של פורמטים תואמים בין החלקים השונים של הקוד.התמודדות: ביצענו ניסויים בשלבים, בדקנו את תקינות החיבורים בנפרד, השתמשנו בלוגים וב-printים לאיתור תקלות בשלב הפיתוח.
2. **קבלת מידע בזמן אמת וניהול עדכונים (MQTT live streaming)**: נדרשה קליטה של נתונים חיה משני חיישנים שונים (indoor ו-outdoor) באמצעות פרוטוקול MQTT. הדבר חייב טיפול יעיל בנתונים מתעדכנים והתמודדות עם אפשרות לשגיאות בזמן ריצה.התמודדות: יצרנו callbacks מובנים ל-on\_connect ו-on\_message, והוספנו טיפול בשגיאות באמצעות try-except כדי למנוע קריסה של המערכת במקרה של שגיאה בנתונים הנכנסים.
3. **שימוש במבנה קוד קריא ושמורכבות נמוכה**: בגלל היקף הפונקציונליות, היה צורך לארגן את הקוד כך שיהיה ברור ונוח לתחזוקה – במיוחד לאור ריבוי מקורות המידע (חיישנים, firebase, נתונים מאתרים).התמודדות: מיקמנו את הפונקציות המרכזיות בצורה מודולרית והקפדנו על שמות משתנים ברורים. השתמשנו בהערות ובפירוק לקבצי קוד לפי תפקידים.
4. **אתגרי ויזואליזציה של נתונים**: מאחר שמדובר בנתונים מתעדכנים, הצגת המידע בצורה ברורה ויזואלית הייתה מאתגרת – במיוחד באקוסיסטם של Jupyter Notebook שבו כל cell מרונדר מחדש.התמודדות: השתמשנו ב־matplotlib וב־ipywidgets כדי לאפשר עדכונים אינטראקטיביים ויצירת ממשק משתמש בסיסי. כמו כן, נעשה שימוש ב־clear\_output כדי למנוע הצפה של מידע מיותר.
5. **אבטחת מידע בחיבור ל-Firebase :** בעת עבודה עם Realtime Database ב-Firebase, יש להתחשב בזכויות גישה ובמפתחות API על מנת להימנע מחשיפה של מידע רגיש.התמודדות: נעשה שימוש ב-credentials וב־firebase admin כדי לוודא שהגישה מתבצעת בצורה מאובטחת ומבוקרת.

**תיק תחזוקה**

| **Home** | מסך ראשי שמאגד את כלל הרכיבים של המערכת ומציג את מצב החיישנים בזמן אמת. מאפשר למנהל לעבור בקלות בין החלקים השונים (Statistics, Search, Game Mode) באמצעות כפתורי ניווט. | render\_sensor\_status() – מציג תיבת מידע מעוצבת עם ערכי החיישנים Indoor ו־Outdoor.  fetch\_current\_sensor\_values () – קורא את הערכים העדכניים של החיישנים ממסד הנתונים (/sensor\_values).  switch\_view() – מחליף את התוכן שמוצג בהתאם ללחיצה בתפריט (לדוגמה: מעבר ל־Search).  admin\_view() – מכיל את כלל התצוגות במסך Home כולל החיישנים והטופס לצפייה בהיסטוריה.  mng\_ui() – טופס צפייה בנתוני היסטוריה לפי טווח תאריכים.  show\_history\_table() – מציג טבלת היסטוריית מדדים לפי תאריכים שנבחרו. |
| --- | --- | --- |
| **Statistics** | הצגת מידע גרפי חי ממסד הנתונים, לצד טבלה סטטיסטית של המדדים – כולל ממוצע, מקסימום, סטיית תקן ואיתור חריגות. | live\_plot() – מריץ גרפים חיים מהחיישנים כולל חישוב אנומליות והצגת טבלה סטטיסטית.  fetch\_sensor\_data(sensor\_type='indoor') – מביא עד 15 קריאות אחרונות מחיישנים לפי Indoor/Outdoor.   create\_stats\_table(data\_dict, thresholds) – מחשב ממוצע, סטיית תקן, מקסימום, מינימום, ואנומליות לכל מדד.   thresholds – מגדיר את טווחי הערכים התקינים (למשל: טמפרטורה בין 20 ל־30 מעלות).   sensor\_dropdown – תפריט בחירה בין סוגי חיישנים.   start\_button – כפתור להפעלת הגרף החי.   graph\_output, table\_output – אזורים גרפיים להצגת גרף וטבלת סטטיסטיקה בהתאמה. |
| **Search** | לאפשר למשתמש לחפש ביטויים מתוך מסמכי אתר mqtt.org. המנוע משתמש באינדקס מילים שנבנה מראש ומחזיר קישורים לפי רלוונטיות. | search\_sentence\_links(sentence) – מחפש ביטוי שלם, מפעיל חיפוש על כל מילה ומדרג תוצאות לפי התאמה.  search\_word(word) – מחפש מופעים של מילה בודדת במסד הנתונים /word\_index.  on\_search\_click(b) – מופעל בלחיצה על כפתור החיפוש ומציג את התוצאות בצורה מעוצבת.  search\_input – תיבת הקלט של המשתמש.  search\_button – כפתור לביצוע החיפוש.  results\_output – תיבת תצוגת התוצאות.  search\_ui – מאגד את כל רכיבי החיפוש למסך אחד. |
| **Game Mode** | לאש]ככ לאפשר למהנדסים לפעול לפי מצבים מצבים הנדסיים שונים (למשל: חריגה בטמפרטורה) באמצעות ממשק המדמה סביבה תעשייתית. כל פעול כל פעולה מקנה ניקוד אישי. | display\_login\_screen() – טופס התחברות עם הזנת מזהה משתמש.  load\_points\_for\_id(user\_id) – טוען את מספר הנקודות הנוכחי של המשתמש ממסד הנתונים.  update\_points\_for\_id(user\_id, points) – שומר את הניקוד החדש של המשתמש תחת מזהה אישי.  display\_game\_mode() – בונה את מסך המשחק שכולל פעולות מערכתיות והצעות לפעולה.  make\_cb() – יוצרת דינמית כפתור, שדה קלט וכפתור אישור לכל פעולה הנדסית (כמו "Adjust Temperature").  FBconn.post('/engineerActions/...') – רושם את הפעולה שבוצעה (כולל ערך, פעולה, ניקוד) במסד.  submit\_sugg.on\_click() – שומר את ההצעות שסומנו (Suggested Actions) ומעדכן את הניקוד.  current\_id – מזהה המשתמש הנוכחי.  current\_points – סך הנקודות של המשתמש. |

**תיק משתמש**

**הסבר כללי על המערכת**

המערכת שפותחה נועדה לאסוף, להציג ולנתח נתונים המתקבלים מחיישנים מסוג Indoor ו־Outdoor, במקביל להפעלת מנוע חיפוש אינטרנטי ואלמנטים של ממשק משחק (Gamemode) עבור משתמשים טכניים. המערכת מספקת למשתמשים תמונה חיה של מצב הסביבה באמצעות נתוני חיישנים, לצד כלים סטטיסטיים מתקדמים, יכולות חיפוש חיצוניות, וחוויית משתמש המדמה משימות מהעולם התפעולי לצורכי תרגול, למידה ואיסוף ניקוד אישי.המערכת בנויה לפי עקרונות Microservices – כל רכיב אחראי לתפקוד עצמאי, עם ממשקי תקשורת ברורים ביניהם. מבנה זה מאפשר גמישות והתאמה נוחה לתחזוקה עתידית או הרחבות מערכת.בכניסה למערכת מוצג מסך הבית הכולל תפריט ניווט ראשי עם ארבעה כפתורים:  
HOME, STATISTICS, GAMEMODE, ו־SEARCH.  
כל כפתור מעביר את המשתמש למסך ייעודי, כשהמעברים מהירים ושומרים על עקביות עיצובית לשם חוויית שימוש רציפה.

**פירוט מסכים**

HOME: במסך הבית מוצגים נתוני החיישנים (INDOOR/OUTDOOR) בזמן אמת.  
בין הנתונים: מרחק, טמפרטורה, לחות, לחץ ועוד.

STATISTICS: במסך זה המשתמש בוחר בין חיישני Indoor או Outdoor. המערכת מציגה גרף בזמן אמת של ערכים שנאספו, יחד עם טבלה סטטיסטית הכוללת ממוצע, סטיית תקן וערכים קיצוניים. המידע כולו מתעדכן בזמן אמת מתוך מסד הנתונים Firebase.

GAMEMODE: המשתמש מזדהה בעזרת מזהה UID ומקבל משימות תחזוקה טכניות המדמות סביבה תפעולית. כל משימה מבוססת על הנתונים הנוכחיים מהחיישנים, וביצועה מזכה בניקוד. הנתונים נשמרים במערכת לצורכי השוואות עתידיות.

SEARCH: מאפשר למשתמש להזין ביטוי חופשי ולקבל קישורים לאתרים שבהם נמצאו התאמות רלוונטיות. מנוע חיפוש פנימי סורק אתרי אינטרנט, יוצר אינדקס הפוך (Inverted Index), ומחזיר תוצאות מדורגות לפי מספר הופעות של מילים רלוונטיות בביטוי.

**מעברים בין מסכים**

המערכת בנויה כך שכל ממשק מוצג במסך ייעודי ונפרד, ומעבר בין מסכים מתבצע באמצעות תפריט ניווט ראשי, הממוקם בראש או במרכז הדף, בהתאם לעיצוב. הכפתורים המוצגים במסך זה מאפשרים ניווט ישיר לארבעה אזורים מרכזיים: Home, Statistics, GameMode, ו-Search.לחיצה על כפתור Home, המשתמש חוזר למסך הראשי של המערכת - שם ניתן לקבל מידע כללי, התמצאות ראשונית ובחירה בפעולה הבאה. לחיצה על כפתור Statistics מעבירה את המשתמש למסך ניתוח החיישנים, שם הוא בוחר את סוג החיישן (indoor או outdoor), מריץ גרף חי של הנתונים, וצופה בטבלת סטטיסטיקה מתקדמת.לחיצה על כפתור GameMode מעבירה את המשתמש למסך ה-Gamemode, ומוביל לאזור המיועד למשתמשים שמעוניינים לבצע סימולציה של משימות תחזוקה, לצבור נקודות ולבצע פעולות בהתאם לנתוני המערכת בפועל. לאחר הזנת UID, המשתמש מקבל משימות המתאימות לו, ויכול לבצע אותן בלחיצת כפתור. בלחיצה על כפתור Search, נפתח מסך החיפוש אשר מאפשר הזנת משפט חופשי בשדה טקסט, מריצה את אלגוריתם החיפוש, ומחזיר קישורים רלוונטיים על פי ניתוח המילים במשפט.

**טעויות אפשריות ומנגנוני טיפול:**

המערכת ערוכה לטיפול במצבים חריגים ושגיאות פרשנות מצד המשתמש, תוך הצגת הודעות ברורות והנחיות לפעולה. להלן דוגמאות עיקריות:

חיפוש שלא הניב תוצאות: אם המשתמש מזין מונח שלא קיים באינדקס, תוצג הודעה בהתאם.

הזדהות שגויה במצב המשחק: אם מוזן UID שגוי או ריק, תוצג הודעה מפורשת עם אפשרות להזדהות מחדש או להירשם.

פעולה ללא נתוני חיישן זמינים: המערכת מונעת ביצוע פעולות שאין עבורן מידע עדכני, ומציגה הודעת הסבר בהתאם.

שגיאות מערכת: במקרה של תקלות מערכתיות (למשל ניתוק מ־Firebase), מוצגת הודעה זמנית עם ניסיון חיבור חוזר אוטומטי. כל הודעות השגיאה נכתבות בשפה ברורה, ללא מונחים טכניים, ומאפשרות למשתמש להמשיך לעבוד בצורה רציפה ובטוחה.

**עליכם להתייחס להצגות בסשן שלכם, ולמלא משוב לפחות ל -5 צוותים ביום בו הצגתם.**

| הערת משוב | האם לדעתכם יש צורך בשינוי במערכת בעקבות ההערה? | נימוק |
| --- | --- | --- |
| את אלמנט המשחקיות שיהיה להכנס עם סיסמא ולא רק תז. | לא | תעודת זהות היא ייחודית לבן אדם לכן אין צורך גם בסיסמא על מנת לשחק. |
| נראות. גופן קטן | כן | שיפרנו את העיצוב. |
| גרפים נראים טוב ולא עמוס | לא | הערה חיובית. |
| אלמנטים שאהבתי במערכת-פיצר של מזג האוויר | לא | הערה חיובית. |

**Code review**

| **הערה** | **תגובה** | **האם נעשה שינוי בקוד?** |
| --- | --- | --- |
| **פשטות**- שינוי סדר הפונקציות בתוך התאים והוספת פונקציות עזר יעזרו בפישוט את הקוד. | הצוות הסכים שבמקום ספציפי יש מקום לשינוי סדר הפונקציות בתא בשביל להקל על הקריאה ובמקום נוסף להוסיף פונקציית עזר על מנת לפשט את הקוד. | כן, שינינו את סדר הפונקציות בבלוק מסוים והוספנו פונקציית עזר על מנת לפשט את הקוד ולארגנו בצורה טובה יותר. |
| **מימוש**-הקוד עומד בדרישות | הצוות הסכים שהקוד עמד בדרישות הנדרשות. | לא, הקוד עמד בדרישות באופן מלא. |
| **מודולריות**- פיצול חלק מהקוד לתאים נפרדים. | הצוות ציין שהם מבינים את החשיבות של מודולריות, ומתכננים לפצל חלק מהקוד לתאים נפרדים. | כן, בכדי שהקוד יהיה קל לתחזוקה ומובן יותר פיצלנו חלק מהפונקציות המרכזיות לתאים נפרדים. חלקים אחרים הפרדנו לתאים שונים, וכך אין כפילות קוד וקל לבצע שינויים בצורה ממוקדת ויעילה. |
| **יעילות**- ניתן להציג את הגרפים של הסטטיסטיקות בצורה יעילה יותר. | הצוות קיבל את ההערה, אך החליטו שהקוד כן עומד בדרישות ובשל כך לא יבצעו שינוי. | לא, לתפיסתנו נתוני הסטטיסטיקות והגרפים מוצגים בצורה ברורה ויעילה, אין שכפול קוד, הגרפים ברורים ונוחים לצפייה ועומדים בדרישות המטלה. |
| **באגים** - לא נצפו באגים | הצוות הסכים כי לא נמצאו באגים. | לא, כי אין באגים. |
| **טיפול בשגיאות**- השגיאות מטופלות היטב. | הצוות הסכים כי טיפול בשגיאות מתבצע כאוי. | לא, הסכמנו שטיפול בשגיאות מתבצע בצורה טובה. |

**מקורות**

פורמטיים עיקריים לכלי AI שעשינו בהם שימוש ChatGPT, Gemini:

1. "למה הקוד שלי לא מייבא נכון את הנתונים מ Firebase?"
2. "איך לשמור נכון את הנתונים מהמעבדה וה־time בתוך Firebase כדי שהקוד יקרא את הנתונים כמו שצריך?"
3. "איך לשלב בצורה הטובה והפשוטה ביותר צ’אט בוט שמתחבר ל־Gemini"
4. "איך לעצב טוב יותר את הגרפים שלי במסך הסטטיסטיקות כך שייראו מודרניים, ברורים ונוחים לקריאה?"
5. "מאיזה API כדאי למשוך תחזית מזג אוויר ל־4 ימים הקרובים? האם OpenWeatherMap מתאים לזה?"
6. "איך להשתמש ב־ipywidgets כדי לעצב Tabs עם ניווט בין מסך מנהל?"
7. "איך להתמודד עם רשומות חסרות או פגומות בבסיס הנתונים ?"

**מקורות נוספים:**

* **Google Colab** הסביבה שבה פותחה כל המערכת. איפשרה שילוב של קוד Python, גרפים, Widgets, טפסים, ואינטגרציה מלאה עם Firebase, Gemini ו־Gradio – בצורה עננית וחינמית.
* **OpenWeatherMap API** שימש לשליפת תחזית מזג האוויר עבור 4 הימים הקרובים, והוזן לעמוד הראשי של המערכת.
* **Google Gemini API** **(צ'אט בוט)** שולב במערכת לצורך מתן מענה חכם למשתמשים בזמן אמת על שאלות כלליות או הקשורות לנתונים.
* **Firebase** שימש כבסיס הנתונים המרכזי (Realtime Database) של המערכת. נשמרו בו ערכי חיישנים בזמן אמת, נתוני היסטוריה לסטטיסטיקות, תוצאות חיפוש, ניקוד גיימיפיקציה ועוד.
* **Gradio** ליצירת ממשקים גרפיים אינטראקטיביים להצגת נתוני חיישנים, גרפים חיים, תיבת צ'אט עם Gemini, תחזית מזג האוויר, מנוע חיפוש וטופסי פעולה. שימש במיוחד בבניית מסך הניהול הראשי.
* **Matplotlib + pandas** ליצירת גרפים ויזואליים של טמפרטורה לפי תאריך, עם תמיכה במיון, תאריכים, סינון לפי מיקום, והצגה נוחה של נתונים שנמשכו מ־Firebase לאחר עיבוד ב־PySpark.
* **Ipywidgets** לסידור ממשק המשתמש בעבודה, כולל חלוקה לטאבים (טאב ניהול, טאב סטטיסטיקות, מנוע חיפוש, צ'אט, גיימיפיקציה), וכן לבחירה דינמית של טווח תאריכים, מיקום (INDOOR/OUTDOOR) וסוג חיישן.