מבוא למחשוב ענן - סמסטר אביב תשפ"ה

תרגיל בית 3

**מגישים: קבוצת "קקדו"**

עומר לב | ת"ז 209403427

מתן ראובן טל | ת"ז 208017772

ספיר גרסטמן | ת"ז 323116186

אריאל לניאדו | ת"ז 318930393

נועה שיטרית | ת"ז 206943219

טל יגודין | ת"ז 324256536

**קישור לעמוד GitHub:**

<https://github.com/Method-for-Software-System-Development/Cloud_Computing>

**מהנדס המערכת:** טל יגודין

**הוראות להפעלת המערכת**

1. **כניסה למחברת הראשית:**

יש לגשת לתיקיית gui ולפתוח את הקובץ dashboard.ipynb.

קישור ישיר: <https://github.com/Method-for-Software-System-Development/Cloud_Computing/blob/main/gui/dashboard.ipynb>

1. **הרצת שלב ההתקנה וההכנה:**

יש להריץ את תא הקוד הראשון תחת הכותרת:SETUP FOR DASHBOARD

שלב זה כולל התקנת ספריות, ייבוא מחברות לוגיות, חיבורים ל- Firebase ועוד...

**התהליך נמשך כ־2 דקות.**

בסיומו, יש לוודא שמתקבל הפלט:

Setup completed successfully.✅

1. **הרצת המערכת:**

יש להריץ את תא הקוד השני. תוך מספר שניות תופיע האפליקציה בתחתית המחברת.

כברירת מחדל, הממשק מוצג בתוך המחברת (בהתאם לדרישה), אך לחוויית שימוש טובה יותר במסך מלא, ניתן לפתוח את המערכת בקישור חיצוני בלבד (ללא הצגה בתוך הקולאב) באמצעות הקוד הבא:

במקום demo.launch() שנמצא בשורה האחרונה בקוד יש לכתוב/להדביק:

demo.launch(share=True, inbrowser=True, inline=False)

**הערה:** לא מומלץ להפעיל את המערכת גם בתוך המחברת וגם בקישור החיצוני במקביל- הדבר עלול לגרום לתקלות בהצגת הנתונים או בקבלת קלט.

1. **נתוני התחברות לבדיקה:**

שם משתמש: matan\_tal

סיסמה: 1234

**באגים ידועים:**

* לעיתים, הטבלאות בסימולטור תקלות (Fault Simulator) אינן נפתחות במלואן עקב מגבלות של Gradio.

אם זה קורה, יש ללחוץ על נקודה כלשהי בטבלה והיא תתרחב בצורה תקינה.

* **מצב כהה (Dark Mode) אינו נתמך כרגע.**

יש לעבור למצב בהיר (Light Mode) בהתאם לצורך- או דרך תפריט ההגדרות של Gradio הממוקם בתחתית הדף, או על ידי שינוי הגדרות התצוגה בדפדפן.

* אם השורה הבאה נמחקה מטעמי אבטחת מידע של GitHub:

os.environ["GEMINI\_KEY"] = "AIzaSyA8TV62T0jHb21JyjbTyRNfUPz7Gr-KQQk"

יש להדביק את מפתח ה- API של Gemini במקום המיועד לכך במהלך שלב ההתקנה.

מפתח זה אחראי על חיבור הצ'אטבוט. ללא הגדרה זו- מודול הצ'אט לא יעבוד כראוי.

**חלק ראשון: חלוקת משימות ובדיקות קבלה**

**להלן חלוקת העבודה בצוות:**

|  |  |  |
| --- | --- | --- |
| שם חבר הצוות ותפקיד בתרגיל זה | משימות שהוקצו | משימות שהושלמו |
| טל יגודין- מהנדס המערכת ו- Scrum Master | * להוביל את תהליך העבודה לפי עקרונות Agile-Scrum. * לתאם בין חברי הצוות ולוודא שכל אחד מבין את משימותיו. * לקבוע ולהנחות פגישות סטטוס קבועות. * לוודא שהתהליך מתנהל בצורה יעילה ושיש עמידה בזמנים. * לשמש כממשק בין הצוות לבין המרצה (במידת הצורך). | המשימות הושלמו בהצלחה. |
| עומר לב- Product Manager | * כתיבת תיק למתכנת. * כתיבת תיק למשתמש. * הכנת סרטון שיווקי. | המשימות הושלמו בהצלחה. |
| מתן ראובן טל- Backend Developer | * מימוש צ'אט-בוט. * סידור Microservices בפרויקט. | המשימות הושלמו בהצלחה. |
| ספיר גרסטמן- QA | * ביצוע בדיקות קבלה (מפורטות בהמשך) | המשימות הושלמו בהצלחה. |
| נועה שיטרית- UI | * שיפור ממשק המשתמש. * הוספת טקסטים והנחיות למשתמש לשיפור שקיפות האלגוריתם. | המשימות הושלמו בהצלחה. |
| אריאל לניאדו- Frontend Developer | * שיפור ממשק המשתמש. * הגדרת KPI במערכת. * ניתוח BigData. | המשימות הושלמו בהצלחה. |

**בנוסף, כל אחד מחברי הצוות היה אחראי על סידור ותיעוד המחברות עליהן עבד בתרגילים הקודמים.**

בדיקות קבלה

|  |  |  |  |
| --- | --- | --- | --- |
| **מספר בדיקה** | **תיאור הבדיקה** | **תנאים מוקדמים** | **תוצאה צפויה** |
| 1 | הרצת המערכת מתוך המחברת (Colab) | פתיחת המחברת הראשית | כל הרכיבים עולים (Gradio UI, טעינת רכיבים, תקשורת תקינה) |
| 2 | בדיקת קליטת מידע מחיישנים פנימיים וחיצוניים | חיבור ל־MQTT או הפעלת סימולציה | הנתונים מוצגים בלייב / מתעדכנים בטבלת DB |
| 3 | בדיקת גילוי תקלות ע"י Fault Controller | סימולציית ערך לא תקין מחיישן או ערכים אמיתיים לא תקינים | תקלה מזוהה ונשמרת במסד הנתונים |
| 4 | תפקוד הצ’אט-בוט | פתיחת ממשק Gradio | המשתמש מקבל תשובה אינטליגנטית ורלוונטית מהצ’אט-בוט |
| 5 | בדיקת תפקוד המערכת הגיימיפיקטיבית (Checklist + ניקוד למהנדסים) | הכנסת תקלה למערכת (גיימיפיקציה) | מוצג Checklist לתיקון התקלה, ולאחר סימון – המהנדס מקבל ניקוד |
| 6 | בדיקת leaderboard | ביצוע פעולה שנותנת ניקוד (תיקון תקלה) | ניקוד נוסף למשתמש וניתן לראות את הניקוד החדש ב leaderboard |
| 7 | בדיקת גרפים – הצגת מידע סטטיסטי מהחיישנים | הפעלת Sensor\_stats | מופיעים גרפים עם המידע מה-DB (לדוג' טמפ', לחות) |
| 8 | בדיקת מנוע חיפוש – איתור מונח רלוונטי | הפעלת Search\_index | מוחזרת תוצאה קרובה למונח שהוזן או שלא קיים במידה ומילה קרובה לא קיימת באינדקס |
| 9 | בדיקת שקיפות אלגוריתמית (הסבר למשתמשים על איך הדברים עובדים) | גישה לממשק המשתמש | מוצג הסבר קצר וקריא על איסוף הנתונים, אלגוריתמים ושימוש במידע |
| 10 | בדיקת אפשרות להריץ את כל המערכת מ־Notebook אחד ללא צורך בשרת Flask או קבצים חיצוניים | פתיחת המחברת בלבד | כל השירותים (כולל ממשק, קלט, עיבוד, תצוגה) זמינים ישירות מתוך המחברת |
| 11 | בדיקה האם המערכת מגיבה מיידית לשינוי במצב (למשל תקלה חדשה, הצגת גרפים או שאלה לצ’אט-בוט) | שינוי פתאומי בערך חיישן / שליחת שאלה לצ’אט / בקשת גרף בזמן ספציפי | המערכת מגיבה בתוך שניות: התקלה נרשמת ומוצגת, או שהתקבלה תגובה חכמה מהצ’אט-בוט / קבלת גרף מתאים |

חלק שני: השלמת בניית המערכת

שימוש ב- **Microservices**

**במערכת שלנו עבדנו עם כמה מחברות שאחראיות על הלוגיקה וכל מחברת היא** microservice **בפני עצמה, להלן פירוט המחברות:**

* **Chatbot\_controller:**

**אחראי על יצירת וניהול התקשורת עם הצ'אט-בוט. מקבל שאילתות ממשתמשים, שולח אותן לעיבוד, ומחזיר תגובות רלוונטיות בהתאם להקשר.**

* **Fault\_controller:**

**מזהה תקלות במערכת באמצעות ניתוח נתוני חיישנים. מאתר ערכים חריגים או לא תקינים, ושומר את המידע במסד הנתונים לצורך תחקור או טיפול עתידי.**

* **Mqqt\_sim\_indoor:**

**אחראי על איסוף נתונים מחיישנים פנימיים (לדוגמה, טמפרטורה או לחות בתוך המעבדה). כולל יכולת סימולציה כאשר החיישן אינו זמין או תקול.**

* **Mqqt\_sim\_outdoor:**

**מבצע תפקיד דומה לזה של המודול הפנימי, אך עבור חיישנים חיצוניים. מאפשר גם סימולציה במקרה של תקלה או חוסר זמינות.**

* **Repair\_controller:**

**אחראי על צד ה"גיימיפיקציה" של המערכת – מציג למשתמשים (מהנדסים) רשימת בדיקה לטיפול בתקלות. עם השלמת הטיפול, המערכת מעניקה ניקוד או דירוג למשתמש שביצע את התיקון.**

* **Search\_index:**

**מנוע חיפוש פנימי המבוסס על אינדקס קיים מראש. מאפשר חיפוש מהיר והתאמת קישורים למידע רלוונטי במערכת.**

* **Sensor\_stats:**

**מייצג נתונים שנאספו מהחיישנים ומאוחסנים במסד הנתונים באמצעות גרפים. מאפשר למשתמשים לצפות במגמות, סטטיסטיקות וניתוחי המידע בזמנים שונים.**

* **User\_controller:**

**אחראי על ניהול משתמשים והתחברות. כולל גם ניהול לוח תוצאות (**Leaderboard**) שמתעדכן על פי פעילות המשתמשים במערכת, כמו תיקון תקלות.**

**בחרנו לחלק את העבודה למספר מחברות (**notebooks**) נפרדות, בשל מגבלה מובנית של** Google Colab**- חוסר היכולת לעבוד במקביל על אותה מחברת על ידי מספר משתתפים. באמצעות הפיצול, יכולנו לפתח מספר חלקים של המערכת בו-זמנית, מבלי להפריע לעבודה של חברי הצוות האחרים.**

**בנוסף, פיצול הפרויקט למספר רכיבי** microservices **אפשר לנו לפתח מודולים שונים בצורה עצמאית, ללא תלות ישירה בפיתוח המודולים האחרים. במקרה בו פונקציונליות מסוימת עדיין לא הייתה מוכנה, יכולנו להיעזר בקריאות ל-** API**, וכך להמשיך את הפיתוח של רכיבים אחרים תוך שמירה על רצף עבודה. יתרון נוסף בגישה זו הוא שלא נתקלנו בהתנגשויות קוד (**merge conflicts**), מה שתורם ליעילות ולסנכרון טוב יותר בין חברי הצוות.**

ה- **KPI** בפרויקט

**ה-** KPI **הרלוונטיים שבאו לידי ביטוי בפרויקט שלנו הינם:**

|  |  |  |  |
| --- | --- | --- | --- |
| **KPI** | **תיאור** | **קשר לדרישות הלא פונקציונליות** | **אופן בדיקה** |
| **System Uptime (%)** | אחוז זמן שהמערכת זמינה | זמינות בזמן אמת | בדיקת זמינות שבועית/חודשית עם כלי ניטור |
| **Alert Response Time** | הזמן בין זיהוי תקלה להצגת התרעה בממשק | נדרש להציג התרעה בתוך שנייה | ביצוע בדיקות עומס למדידת השהייה בזמן אמת |
| **Data Update Latency** | זמן מרגע קבלת נתון מחיישן עד שהוא מוצג בדשבורד | עדכון רציף של נתוני סטטוס כל 5 שניות | סימולציה של נתונים ממקור חיצוני ומדידת זמן העדכון |
| **Scoring Transparency Rate** | אחוז המשימות בהן מוצג פירוט ניקוד ברור למשתמש | שקיפות בניקוד | סקר משתמשים + בדיקת ממשק הצגת ניקוד בפועל |
| **Historical Data Availability** | אחוז השלמות של נתוני עבר בגרפים ובדוחות | יכולת השוואת ביצועים היסטוריים | השוואה בין גרפים קיימים לנתוני אמת |
| **System Extensibility Score** | מספר שדות נתונים חדשים שניתן להוסיף מבלי לשנות מבנה בסיס נתונים | בדיקות הוספת שדות חדשים בממשק ובמאגר נתונים | בדיקות תוספת שדות חדשים בממשק ובמסד נתונים |
| **Bug Resolution Time** | הזמן הממוצע לטיפול בתקלה שהתגלתה | מושפע מממשק התקלות ומנגנון הניקוד | מדידת זמן בין זיהוי תקלה לביצוע משימה לתיקון התקלה |
| **User Satisfaction Index (USI)** | ציון שביעות רצון המשתמשים מהממשק והביצועים | תומך בכל הדרישות שממוקדות בחוויית המשתמש | סקרי משתמשים |
| **Leaderboard Engagement Index** | מעיד על תמרוץ המשתמשים דרך ניקוד מצטבר | מעודד שיפור מתמיד באמצעות תחרות בין המשתמשים וחיזוק תחושת הישג | חישוב ניקוד ביחס לתקלות שהמשתמש פתר ודירוג לארוך זמן |
| **Sensor Data Continuity** | שלמות הנתונים המתקבלים מהחיישנים ברציפות | זיהוי תקלות תקשורת ובדיקת עקביות הנתונים | הצגת הנתונים בגרף (חורים במידה ולא מתקבלים נתונים) |
| **Search Index Effectiveness** | רלוונטיות התוצאות החיפוש באתר   MQTT | מאפשר גישה נוחה ומהירה למידע | קישור לאתר ואת מספר מופעי החיפוש בו |

שקיפות אלגוריתמית בפרויקט

**במערכת שלנו שמנו דגש על שקיפות אלגוריתמית ברמת הצגת הנתונים וגם באופן פעולת המערכת.**

**האופנים שבן השקיפות באה לידי ביטוי הינם:**

1. מדריך מפורט למשתמש:

**למשתמש מוסברים הדברים הבאים:**

* + **הסבר כללי למערכת, מה כולל כל אזור במערכת, מה מטרתו ואילו פעולות המשתמש יכול לבצע בו.**
  + **מה הם החיישנים הקיימים במערכת, כיצד נמדדים.**
  + **כיצד ניתן לצבור ניקוד, כולל בתוכו הסבר על תהליך הסימולציה ומה התנאים לקבלת ניקוד ומציג כי קיים יחס בין הניקוד לבין הזמן שלוקח לטפל בבעיה.**
  + **אזור** FAQ **בו מוצגות שאלות נפוצות לגבי שימוש במערכת ותשובות.**

1. הצגת מידע גולמי בזמן אמת:

**ב-** Sensor Dashboard **מוצגים למשתמש הנתונים שנשלפים מהחיישנים בזמן אמת, ללא עיבוד מאחורי הקלעים, זה מאפשר למשתמש להבין את הנתונים עליהם מבוססת פעילות המערכת.**

1. תהליך תיקון מונחה:

**בתהליך התיקון תקלה של המשתמש מוצגים למשתמש השלבים המוגדרים שהוא חייב לבצע כדי להשלים את התיקון, כל פעולה מלווה בהסבר ברור.**

**המשתמש רואה את תנאי הסיום בזמן אמת כיוון שרק לאחר סימון כלל הפעולות ניתן להשלים את הטיפול.**

1. לוח ניקוד גלוי:

**המשתמשים יכולים לראות את הציונים של עצמם ושל אחרים בלוח הדירוג, כלל שם וניקוד. הצגת הלוח מחזקת את תחושת ההוגנות והשקיפות בתמרוץ החישוב התקלות שמטופלות.**

1. שקיפות באירועי תקלה:

**הטבלה התקלות הפעילות מוצג למשתמש כלל הנתונים הרלוונטיים להבנת התקלה.**

**הנוסף ניתן למיין את התקלות על פי קריטריונים כמו זמן, רמת קושי (הכולל בתוכו סידור לפי ניקוד) ולפי סוג מיקום החיישן. הצגה זו מאפשרת למשתמש להבין את אופן הסיווג, אופן הזיהוי ומאפשרת לו לסדר את התקלה שמעוניין לטפל בהם על פי סדר העדיפויות הנוח לו, מה שמחזק את השקיפות בתהליך קבלת ההחלטות של המערכת.**

**BigData**

**בגיט תחת תיקיית** logic **מופיעה מחברת בשם** Bigdata\_analysis.ipnyb **המכילה ניתוח לפי מודל** MapReduce **לכלל הנתונים המוחזקים ב-** Firebase**.**

ניתוח נתונים לפי מודל **MapReduce**:

Map- **שלב בו כל הנתונים עוברים מיפוי, כל רשומה נבחנת וממה מופקים זוגות של מפתח וערך.**

Reduce**- שלב בו כל הערכים מאוגדים לפי מפתח משותף ומבוצעת עליהם פעולה מתמטית, במקרה שלנו השתמשנו בממוצע.**

**השיטה מאפשרת עיבוד מקבילי יעיל של כמויות נתונים עצומות ומיושמת באמצעות** Spark**.**

**גרף עבור ניתוח הנתונים ע"י שימוש ב-** MapReduce **ולאחריו מוצג ניתוח כלל הנתונים מכלל הגרפים:**

|  |  |
| --- | --- |
| **עבור Temperature:** | **עבור Humidity:** |
|  |  |

**נוסיף תמונה עבור** MapReduce **בה ניתן לראות את הערכים המופקים עבור כלל הנתונים ב-** DB**:**

תמונה שמכילה טקסט, צילום מסך, תרשים

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**תמונה שמכילה טקסט, צילום מסך, תרשים

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**

**מסקנות העולות מהנתונים שעלו מניתוח הנתונים ע"י** MapReduce**:**

1. **נאספו יותר נתונים מהחיישנים הפנימיים לעומת החיישנים החיצוניים, מה שמעיד על זמינות גבוהה יותר של החיישנים הפנימיים.**
2. **הפער בין הטמפרטורה המקסימלית למינימלית בפנים מעיד על יציבות תרמית.**
3. **תנודות הלחות בפנים גבוהות יותר מאשר בחוץ מה שמעיד על תנאים פחות מבוקרים.**
4. **טווח המרחק גדול בין 210 ל-1730 מה שמעיד על תנועת עצמים במרחב.**
5. **עוצמת האור נעה בין ערכים של 108 ל-3330 מה שמעיד הנתונים מחזיקים מכילים אור יום ולילה, מה שמספק לנו את ההפרשים בין עוצמת התאורה.**

אתגרים בפרויקט

**אתגרים שהתמודדנו איתם במהלך העבודה:**

1. **המעבדה בשבוע 4 הייתה לא ברורה ולא הצלחנו להבין איזה חיישנים עומדים לרשותנו עד לשימוש ב-** API **על מנת לקרוא את המידע מ-** MQTT**.**
2. **תחילה החיפוש** MQTT **נשמר לא על פי המבנה המבוקש במאגר נתונים, והמידע שהוצג לא טעם את הדרישות, לאחר התייעצות עם חברי צוות אחרים מהלימודים הבנו כי המבנה אינו נכון וביצענו שינויים.**
3. **הייתה התלבטות צוותית כיצד להתמודד עם כך שיש מקרים בהם מקבלים מהחיישנים** N/A**, הוחלט בסוף לבחור להגדיר את הנתון במקום** N/A **בתור** None **על מנת שבגרף יוצגו חורים בהם לא מתקבל מידע מהחיישן.**
4. **התקשינו עם העבודה בקולאב כקבוצה גדולה של כשישה אנשים, גוגל קולאב אינו מאפשר עבודה על אותה מחברת במקביל של שני משתמשים. לכן הוחלט כי יש מחברת ראשית שעובדת מול מאגר הנתונים וכל פעולה מקבלה** cell **בנפרד, כל זוג עבד על עמוד נפרד ובכך חילקנו את העבודה למחברות נפרדות ובכך התאפשר לנו לעבוד במקביל.**

**לקראת ההגשה עבדנו יחד חברי הצוות על חיבורי המחברות של ה-** UI **כך שהכל יעבוד תחת אותה מחברת בהרצה סופית, סידרנו באגים בחיבור והחלטנו לשמור על מחברות נפרדות עבור פונקציונליות שונה.**

1. **התקשינו כקבוצה לעבוד עם ספריית** Gradio **זוהי ספרייה שאינה מוכרת לנו והפונקציונליות של הספרייה שונה ממה שלמדנו במהלך התואר, תחילה חלק מחברי הצוות ניסו לעבוד עם** Gradio **והתמקצעו ככל האפשר ולאחר מכן לימדו את שאר חברי הקבוצה.**

**בנוסף לכך למדנו כי הספרייה מייצרת דברים בסיסיים ונצטרך לשכלל את התבנית הבסיסית על מנת לשנות את העיצוב שיהיה מתאים לפרויקט שלנו.**

1. **בתחילת העבודה ניסינו לקרוא נתונים מהחיישנים אך ללא הצלחה, לאחר מכן הבנו כי החיישנים לא עבדו ועל מנת להתקדם בפרויקט היה עלינו לעבוד עם מידע פיקטיבי.**

**בנוסף לכך לא הוצג מידע עבור כל חיישן שמציג מצג של שגיאה במערכת ולכן היה עלינו ללמוד ערכים תקינים שמגיעים מהחיישן ולפי המידע שלמדנו היינו צריכים לייצר קובץ שגיאות הגיוניות לפי טווחי ערכים.**

1. **החלטנו לבנות קובץ בשם** CONTROLLER\_FAULT **על מנת למנוע ריבוי בדיקות לפי סוג החיישן לכל טווח ואז לייצר את השגיאה המתאימה לערך המתאים, לכן הקובץ מציג מבנה ברור עבור כל שגיאה עם ערכים ספציפיים** min, max **עבורם נבדקים ערכים לפי החיישן המתאים ולמיקומו. תפקיד הקובץ לספק ערכים עבור השגיאה המתאימה.**

**חלק שלישי: סגירת הפרויקט**

**תיק למתכנת - מערכת OptiLine**

**קבצי המערכת ופונקציות מרכזיות:**

* **תיקיית firebase- שכבת גישה לנתונים (Data Access Layer)**

**Firebase.ipynb:** אחראי על אינטראקציה עם מסד הנתונים בענן (Firebase)- ניהול משתמשים, חיישנים, תקלות וניקוד.

**שימוש בספריית requests**

האינטראקציה מבוצעת בעזרת REST API. לשם כך נעשה שימוש בספריית requests - ספרייה חיצונית בפייתון שמאפשרת שליחה פשוטה של בקשות HTTP (כגון GET, POST, PUT, PATCH, DELETE) מול שרתים.

הספרייה מטפלת בכל אחת מהפעולות:

* יצירה (Create)
* קריאה (Read)
* עדכון (Update)
* מחיקה (Delete)

פעולות HTTP בשימוש (כולל דוגמה):

1. GET - שליפת נתונים - משמשת לשליפת פרטי משתמשים, תקלות, או נתוני חיישנים.

r = requests.get(f"{firebase\_url}users/{username}.json")

2. PUT - יצירה/דריסה של אובייקט שלם - יוצר או מחליף אובייקט תחת מזהה ידוע.

r = requests.put(

f"{firebase\_url}users/{key}.json",

data=json.dumps(user\_data))

3. PATCH - עדכון חלקי - משמש לעדכון של שדות מסוימים בלבד, מבלי להחליף את כל האובייקט.

r = requests.patch(

f"{firebase\_url}users/{username}.json",

data=json.dumps(update\_data))

4. DELETE - מחיקת נתון - משמש למחיקת משתמש או תקלה.

r = requests.delete(f"{firebase\_url}users/{username}.json")

טיפים לפרקטיקה נכונה עם requests:

* תמיד בדוק את r.status\_code כדי לוודא הצלחה (200, 201).
* קבל את המידע שחזר בעזרת r.json או r.text.
* מומלץ להוסיף טיפול בשגיאות (try/except) סביב r.json.

**שימוש בספריית json**

ספריית json היא ספרייה סטנדרטית (built-in) בפייתון, המשמשת להמרה (serialization/deserialization) בין מבני נתונים של פייתון (dict, list וכו') לבין מחרוזות JSON – הפורמט שבו Firebase Realtime Database מקבל ושולח מידע.

במערכת שלנו נעשה שימוש בפקודה json.dumps לצורך המרה של מילונים (dict) למחרוזות JSON תקניות. השימוש ב- dumps מאפשר שליחה פשוטה ובטוחה של אובייקטים מורכבים כמו נתוני משתמשים, חיישנים ותקלות.

לצורך הפעולה ההפוכה - כלומר, המרת תגובת JSON מהשרת חזרה למבנה נתונים בפייתון - קיימת הפקודה json.loads. עם זאת, בפרויקט זה לא נעשה בה שימוש ישיר, מכיוון שספריית requests כוללת מתודה נוחה בשם r.json אשר מבצעת את ההמרה הזו אוטומטית ומחזירה את הנתונים במבנה של מילון (dict) מוכן לעיבוד.

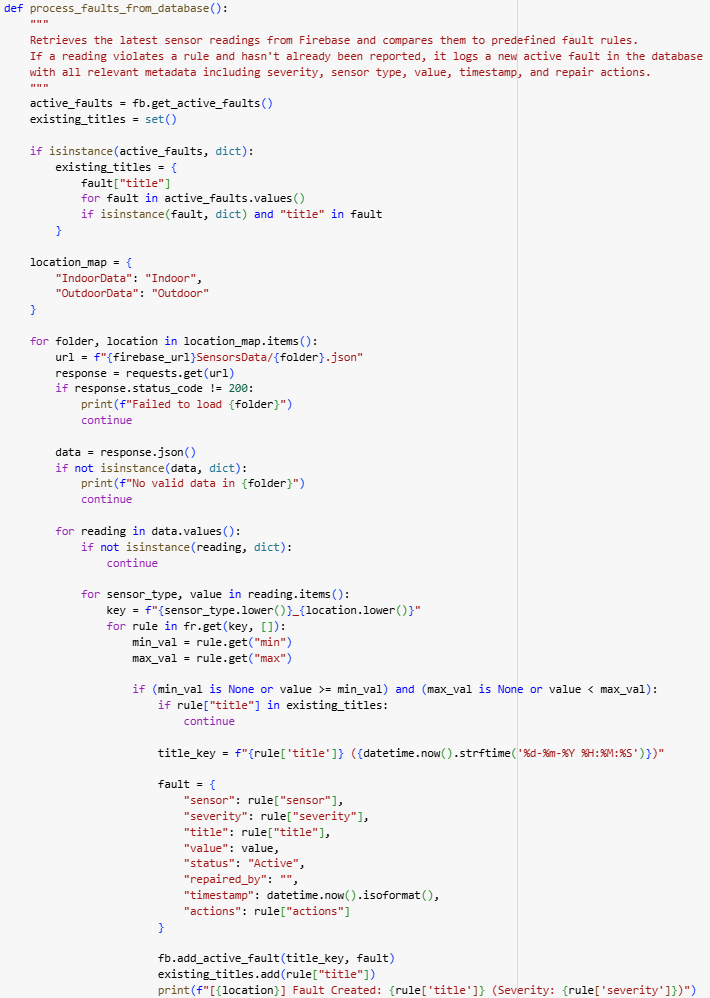
* **תיקיית logic- שכבת הלוגיקה העסקית (Business Logic Layer)**

**1. Fault\_controller.ipynb:**

מכילה את הלוגיקה לזיהוי תקלות, שליפת תקלות פעילות מהמערכת, ועדכון סטטוס של תקלה קיימת. משמשת את הסימולטור להצגת תקלות חיות ומעקב אחר הטיפול בהן.

**הפונקציה process\_faults\_from\_database**

הפונקציה אחראית על זיהוי תקלות חיות במערכת באמצעות שליפה של נתוני סנסורים מ- Firebase והשוואתם ל- "חוקי תקלה" המוגדרים מראש ב- FAULT\_RULES. היא סורקת את נתוני הסנסורים מכל אחת מהקטגוריות (Indoor/Outdoor), בודקת האם ערכים מסוימים חורגים מהטווחים התקניים, ומוודאת שהתקלה אינה קיימת כבר במסד הנתונים. לצורך מניעת כפילויות, נעשה שימוש במבנה set שמכיל את כותרות התקלות שכבר דווחו. בעת גילוי תקלה חדשה, הפונקציה יוצרת מזהה ייחודי (ID) לתקלה על בסיס כותרת התקלה והשעה הנוכחית (datetime.now().strftime), ובונה אובייקט JSON המכיל את פרטי התקלה כולל חומרה, ערך, פעולות תיקון וזמן. התקלה נרשמת למסד הנתונים באמצעות קריאת requests.put דרך מחלקת הגישה ל- Firebase. הפונקציה ממומשת באופן גנרי ותומכת בהוספת חוקים חדשים ללא שינוי מבני בקוד.



**2. repair\_controller.ipynb:**

אחראית על לוגיקת התיקון: התחלת תהליך תיקון, סימון שלבים כבוצעו, סיום תיקון וחישוב ניקוד בהתאם למהירות התגובה וחומרת התקלה.

הקובץ כולל שלוש פונקציות עיקריות:

**1. start\_repair(sensor: str)**

מה עושה:

מתחילה את תהליך התיקון עבור חיישן מסוים, ומעדכנת את הסטטוס במסד הנתונים ל־"Repair Started" יחד עם חותמת זמן (repair\_start\_time).

דגשים טכניים:

* + משתמשת ב־datetime.now(pytz.UTC) כדי לתעד זמן אוניברסלי.
  + מייצרת תצוגה ידידותית לזמן לפי שעון ישראל (Asia/Jerusalem) בפורמט "יום/חודש, שעה:דקה".

**2. calculate\_fault\_score(fault)**

מה עושה:

מחשבת את הניקוד שהמשתמש יקבל על תיקון התקלה, בהתבסס על חומרת התקלה ומהירות התגובה.

דגשים טכניים:

* + ניקוד בסיסי לפי חומרה: Low = 50, Medium = 100, High = 200.
  + בונוס נוסף מחושב לפי משך זמן התיקון – ככל שהתיקון מהיר יותר, כך הבונוס גבוה יותר (מוגבל ל־60 נקודות).
  + הניקוד הכולל מעוגל ומוחזר כערך מספרי.

**3. complete\_repair(sensor: str)**

מה עושה:

מסיימת את תהליך התיקון של תקלה, מחשבת את הניקוד הסופי, מסמנת את התקלה כ"פתורה", ומעדכנת את ניקוד המשתמש.

מהלך הפעולה:

* + מאחזרת את המשתמש הנוכחי המחובר (באמצעות uc.get\_current\_user).
  + מחשבת ניקוד בעזרת calculate\_fault\_score.
  + מעדכנת את התקלה כ"פתורה" ומשייכת אותה למשתמש.
  + מוסיפה את הניקוד למשתמש (fb.update\_user\_score).
  + מחזירה הודעת סיכום הכוללת את כמות ה- XP שהתווספה והניקוד הכולל הנוכחי.

**3. FAULT\_RULES.ipynb:**

כוללת את חוקי הזיהוי והאבחון של תקלות על סמך קריטריונים המוגדרים (כגון סף טמפרטורה, לחות וכו'). מהווה מנוע כללים המייצר תקלות לפי נתוני סנסורים.

**4. mqqt\_sim\_indoor.ipynb / mqqt\_sim\_outdoor.ipynb:**

שתי מחברות המדמות את זרימת המידע מחיישנים פנימיים וחיצוניים (MQTT). מספקות מקור נתונים חי או מדומה לסנסורים כמו טמפרטורה, לחות ולחץ.

**MQTT**

MQTT (Message Queuing Telemetry Transport) הוא פרוטוקול תקשורת קל משקל מבוסס publish-subscribe, המיועד במיוחד להעברת נתונים בין מכשירים באינטרנט של הדברים (IoT). הוא פועל על בסיס שרת מתווך (broker) שאליו מכשירים יכולים לפרסם נתונים (publish) או להאזין להם (subscribe). הפרוטוקול חסכוני ברוחב פס ומתאים במיוחד לסביבות עם קישוריות לא יציבה או דרישות זמן אמת.

**הסבר הפונקציות במחברות:**

**get\_live\_data\_stream(mode="mqtt")**

פונקציה ראשית שנחשפת החוצה (נקראת ע"י ה- dashboard) ומחזירה generator של נתוני סנסור - לפי mode.

* + simulation מחזיר נתונים מדומים (לבדיקות).
  + mqtt מאזין ל- MQTT ומחזיר נתונים חיים מהתור (queue).

**simulate\_data\_stream(delay\_seconds=1)**

יוצר נתונים אקראיים (טמפרטורה, לחות, לחץ) כל שנייה. מאפשר להריץ את המערכת גם ללא MQTT פעיל.

**mqtt\_data\_stream**

מפעיל את מאזין ה- MQTT ברקע (\_start\_mqtt\_listener).

ממתין בתור (queue.get) לנתונים חיים ומחזיר אותם.

**start\_mqtt\_listener**

יוצר לקוח MQTT ונרשם ל- topic (indoor/outdoor) מתוך broker ציבורי.

מפעיל client.loop\_forever בחוט נפרד כדי לא לחסום את הקוד הראשי.

**on\_mqtt\_message**

callback שמופעל כאשר מגיעה הודעה ל- MQTT client.

מנתחת את ה- payload ל- JSON, ומכניס אותו לתור data\_queue.

**5. sensors\_stats.ipynb:**

מטפלת בניתוח סטטיסטי של נתוני סנסורים: שליפה היסטורית לפי תאריך ושעה, חישוב ממוצעים והכנה להצגה גרפית בממשק המשתמש.

הנתונים ממוזגים ממקורות Indoor ו- Outdoor, וממוינים לפי חותמות זמן. הפונקציה המרכזית בקובץ היא fetch\_data\_from\_firebase, אשר מקבלת מחרוזת זמן התחלה (YYYY-MM-DD HH:MM:SS) ומבצעת שליפה של כלל הקריאות מהשרת בטווח של שעה אחת בלבד. היא מטפלת באיחוד של מבני נתונים כפולים, מחליפה ערכים חסרים ('N/A') בערכי None לצורך ניתוח נוח, ומבצעת המרה ל- pandas.DataFrame ממוין לפי חותמות זמן תקניות. באמצעות שילוב של pandas ו- matplotlib, נבנים גרפים ייעודיים לכל פרמטר סביבתי (טמפרטורה, לחות, לחץ, אור, מרחק). הקובץ כולל פונקציות עזר להפקת רשימות תאריכים, חודשים ושנים זמינים במסד הנתונים, וכן מיפוי בין תאריך לרשימת שעות פעילות. המבנה הגמיש של הפונקציות מאפשר אינטגרציה חלקה עם ממשק בחירת טווחי זמן בדשבורד.

**6. search\_index.ipynb:**

מממשת מנוע חיפוש מבוסס אינדקס: יצירת אינדקס ממסמכי mqtt.org, טיפול ב-stop words ו- stemming, ואחזור קישורים רלוונטיים לפי שאילתא.

הקובץ search\_index מממש את מנוע החיפוש עצמו. המנוע נשען על אינדקס מילים המאוחסן ב- Firebase ומכיל עבור כל מילה את רשימת הדפים שבהם הופיעה (DocIDs) ואת מספר ההופעות שלה בכל דף (DocCounts). הפונקציה המרכזית search\_words מפרקת את שאילתת המשתמש למילים, מבצעת תהליך stemming (גזירה לשורש) ומחזירה את הקישורים הרלוונטיים ביותר בהתאם לשכיחותם. במקרה של מילה שלא נמצאה, מתבצע ניסיון להציע תיקון שגיאת כתיב בעזרת difflib.get\_close\_matches. התוצאות מוצגות בפורמט HTML עם קישורים פעילים, המאפשרים גישה נוחה לדפים הרלוונטיים ביותר.

התוכן המאוחסן במסד הנתונים נוצר מראש באמצעות הרצת הקוד שבמחברת IndexCreation, אשר סורקת את האתר באופן רקורסיבי, שולפת את התוכן מכל עמוד HTML, מבצעת ניקוי של stop words, גזירה לשורשים (Porter stemming), וסינון מילים נדירות. לבסוף, הנתונים מועלים למבנה אחיד ומועברים ל- Firebase לשימוש שוטף במנוע החיפוש. תהליך זה מתבצע פעם אחת בלבד (offline indexing), כך שמנוע החיפוש נשאר קל ויעיל בזמן ריצה. המבנה כולו מאפשר הרחבה עתידית לאינדוקס של מקורות נוספים.

**7. user\_controller.ipynb:**

אחראית על טיפול בנתוני משתמשים ברמה הלוגית: התחברות, שליפת נתונים אישיים, גישה לאלפון, עדכון ניקוד ותפקידים.

הפונקציה login מנסה תחילה גישה ישירה לפי מפתח, ובמקרה הצורך סורקת את כלל המשתמשים לצורך אימות. שם המשתמש המחובר נשמר כמשתנה גלובלי current\_user, אשר מאפשר לפונקציות אחרות כמו get\_current\_user או get\_leaderboard לשלוף מידע בהתאם. הפונקציה get\_leaderboard יוצרת טבלת מובילים מסודרת לפי ניקוד, ומוודאת שהמשתמש הנוכחי יופיע בתחתית הרשימה גם אם הוא מחוץ ל- Top 5. הקובץ מיישם "מצב גלובלי" פשוט ונגיש, המתאים למערכות שבהן רק משתמש אחד פעיל בכל רגע נתון.

**8. chatbot\_controller.ipynb:**

הקובץ אחראי על הפעלת העוזר הווירטואלי OptiBot, המשולב במערכת OptiLine.

**הפונקציה ask\_optibot**

הפונקציה מקבלת שאלה מהמשתמש ומנהלת את תהליך התגובה: היא מנתחת את תוכן השאלה כדי לזהות האם היא דורשת מידע תפעולי עדכני (כגון טמפרטורה או תקלות), מבצעת שליפה חיה מהמערכת (למשל דרך מודולי MQTT או Firebase), ומרכיבה prompt מותאם אישית שנשלח ל- Gemini API של Google לצורך הפקת תשובה מבוססת בינה מלאכותית. הפלט חוזר למשתמש בצורת תשובת טקסט אינפורמטיבית באנגלית, המשלבת ידע כללי עם מידע חי רלוונטי. המודול ממומש כ- backend בלבד, ללא לוגיקה של תצוגה, ומשתלב בקלות עם ממשק המשתמש של Gradio דרך היסטוריית שיחה (chat history). המבנה מאפשר הרחבה עתידית לזיהוי הקשרים נוספים, אינטגרציה עם מקורות מידע נוספים, ושיפור חוויית המשתמש בשיח עם המערכת.



* **תיקיית gui- שכבת ההצגה (Presentation Layer)**

**dashboard.ipynb:** מחברת זו מרכזת את כל המחברות האחרות, בונה את הממשק הגרפי בעזרת Gradio, ומספקת למשתמש גישה נוחה לפיצ'רים כמו ניתוח סטטיסטי, ניהול תקלות, מנוע חיפוש, Leaderboard ועוד.

המעבר בין הפאנלים נשלט באמצעות כפתורים ואירועים מוגדרים, כאשר כל פעולה משנה את רכיבי התצוגה הפעילים באמצעות gr.update.

פונקציית card\_streamer מדגימה שימוש ב- generator להזרמת נתוני חיישנים חיים, ופאנל סימולציית התקלות כולל לוגיקה מתקדמת לזיהוי תקלות פעילות, הפעלת טיימר תיקון, ניהול משימות (checkboxes), חישוב ניקוד ותיוג חיישנים.

הקובץ מהווה את נקודת החיבור המרכזית של כלל רכיבי הפרויקט- הוא מפעיל, מציג, מתזמן ומעדכן את המידע באופן אינטראקטיבי, והוא בנוי כך שתומך בהרחבה עתידית של מודולים ותכונות נוספות.

about.md, user\_guide.md: קבצים סטטיים בפורמט Markdown, המכילים טקסטים קבועים כגון תיאור כללי של המערכת ומדריך למשתמש. השימוש בהם מאפשר הפרדת תוכן תיאורי מהקוד, ומסייע בשמירה על קוד קריא ויעיל בתוך המחברת הראשית.

**שימוש בתבניות עיצוב בפרויקט:**

במערכת OptiLine נעשה שימוש בשתי תבניות עיצוב קלאסיות לצורך יצירת ארכיטקטורה מודולרית, קריאה ונוחה לתחזוקה. להלן פירוט התבניות שהוטמעו בפועל:

**Facade - תבנית חזית**

**הסבר על התבנית:**

תבנית Facade מספקת ממשק אחיד וברמה גבוהה עבור מערכת מורכבת או מספר רכיבים פנימיים, תוך הסתרת פרטי המימוש. מטרתה היא להקל על צרכנים חיצוניים להשתמש במערכת מבלי להבין את כל מרכיביה.

**מימוש בפרויקט:**

בקובץ firebase.py מומשה שכבת חזית (Facade) לכל פעולות העבודה מול Firebase - שליפה, עדכון, יצירה ומחיקה של נתונים. פונקציות כמו get\_user, update\_fault ו-add\_active\_fault מסתירות את פרטי קריאות ה- HTTP ו- JSON מאחורי API פשוט, אשר מאפשר לשאר חלקי המערכת לפעול בצורה ברורה ועקבית מול בסיס הנתונים.

**Strategy - תבנית אסטרטגיה**

**הסבר על התבנית:**

תבנית Strategy מפרידה בין האלגוריתם לבין הקוד שמשתמש בו, כך שניתן להחליף את דרך הפעולה מבלי לשנות את הקוד הקורא. האלגוריתמים ניתנים להחלפה דינמית על פי צורך.

**מימוש בפרויקט:**

בפונקציה calculate\_fault\_score שבקובץ repair\_controller.py, מימשנו את תבנית Strategy באמצעות חישוב של ניקוד מבוסס חומרת התקלה ומהירות התגובה. האלגוריתם מופרד משאר רכיבי המערכת, וניתן להחלפה בעתיד (למשל, עבור סוגי משתמשים שונים או קריטריונים מתקדמים יותר) מבלי לשנות את הלוגיקה העוטפת.

**תיק למשתמש - מערכת OptiLine**

**הסבר כללי על המערכת:**

OptiLine היא מערכת דשבורד מבוססת ענן, המיועדת למהנדסים וסטודנטים במעבדת ה-CIM ורובוטיקה של המכללה האקדמית להנדסה בראודה. מטרת המערכת היא לספק ממשק מקצועי ונוח למעקב עבור ניתוח והתנסות בנתוני חיישנים המותקנים בסביבת פס הייצור האוטונומי.

המערכת מציגה נתוני חיישנים בזמן אמת או בסביבת סימולציה, מאפשרת ניתוח סטטיסטי של נתונים היסטוריים, כוללת עוזר חכם מבוסס בינה מלאכותית, מנוע חיפוש טכני, סימולטור תקלות המהווה רכיב גיימיפיקציה המעודד למידה והתייעלות תפעולית.

הגישה למערכת מתבצעת באמצעות התחברות מאובטחת, והיא נגישה מכל מכשיר תומך דפדפן.

**פירוט מסכים:**

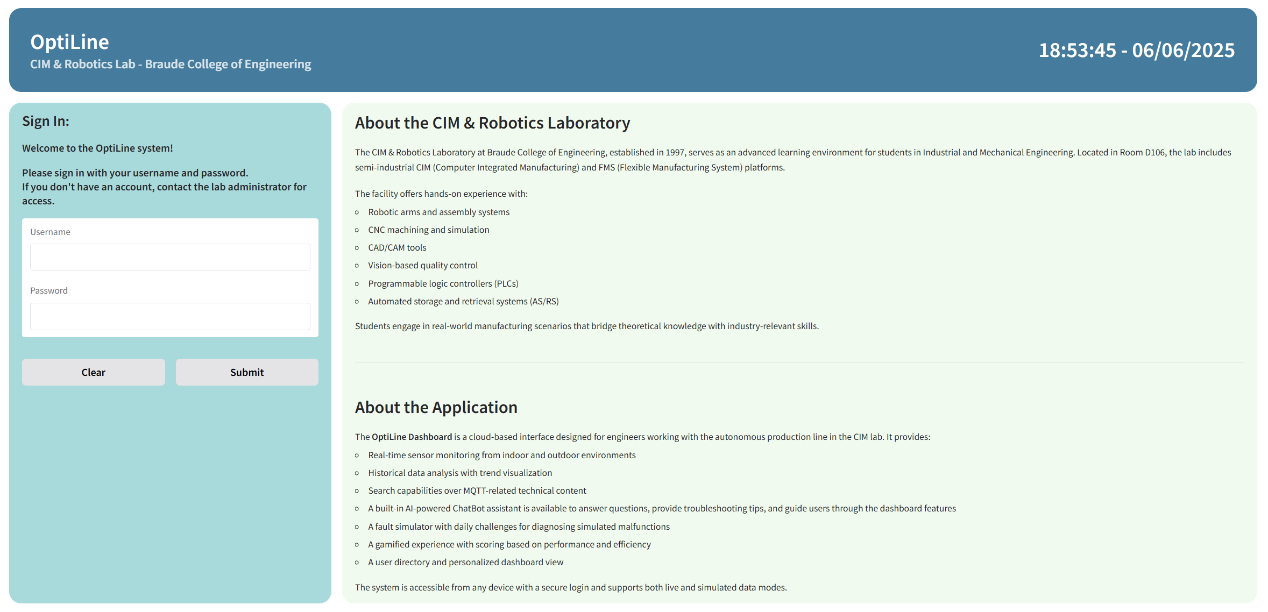
**אלמנטים קבועים בממשק:**

בראש כל מסך מופיע Header קבוע הכולל את התאריך והשעה הנוכחיים.

לאחר התחברות, מופיעים בכל מסך בצדו השמאלי:

* תפריט ניווט צדדי לגישה מהירה לכל המסכים
* Leaderboard המציג את ניקוד המשתמש ומתעדכן לאחר כל אתגר

1. **מסך פתיחה והתחברות:**

****

המסך הראשון במערכת. המשתמש מתבקש להזין שם משתמש וסיסמה.

לאחר התחברות תקינה:

* מוצגים התפריט הצדדי ולוח הניקוד
* מתבצעת הפניה אוטומטית אל מסך ברירת המחדל- תצוגת חיישנים

1. **מסך תצוגת חיישנים (Sensors Dashboard):**

**תמונה שמכילה טקסט, צילום מסך, מספר, תוכנה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**

זהו מסך ברירת המחדל אליו מועבר המשתמש לאחר התחברות.

המסך מציג מדדים חיים (או מסומלצים, בהתאם למצב המערכת) המגיעים מהחיישנים הממוקמים בסביבת הייצור- פנים (Indoor) ו- חוץ (Outdoor).

חיישנים פנימיים:

* טמפרטורה- ביחידות °C
* לחות- באחוזים %
* לחץ אוויר- ביחידות hPa

חיישנים חיצוניים:

* טמפרטורה- ביחידות °C
* לחות- באחוזים %
* עוצמת אור (Dlight)- ביחידות Lux

הערכים מוצגים בתוך מלבנים צבעוניים, המתעדכנים אוטומטית בכל שנייה.

הצגת הנתונים מותאמת לשני מצבים:

* Live- הנתונים מתקבלים בזמן אמת מהחיישנים הפעילים במעבדה.
* Simulation- הנתונים מוזרמים ממקור סימולציה פנימי המדמה תרחישי אמת.

1. **מסך מדריך משתמש (User Guide):**

**תמונה שמכילה טקסט, צילום מסך, תוכנה, מספר

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**

מסך זה מהווה מדריך מקוצר לשימוש במערכת וכולל:

* הסבר על כל רכיבי הדשבורד ומטרתם
* טבלת החיישנים הנתמכים במערכת, כולל יחידות מדידה ומיקום פיזי
* הסבר על שיטת צבירת נקודות ואתגרי התקלות
* קובץ שאלות נפוצות (FAQ) על תפקוד המערכת, כולל מצבי פעולה, ניקוד, גישה לתקלות ועוד

1. **מסך עוזר חכם (OptiLine ChatBot):**

**תמונה שמכילה טקסט, צילום מסך, תוכנה, תצוגה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**

רכיב אינטראקטיבי מבוסס בינה מלאכותית (Google Gemini), המאפשר שיח פתוח עם המשתמש לצורך קבלת מידע והכוונה.

יכולות מרכזיות:

* הסברים על תפעול המערכת: אפשר לשאול כיצד להשתמש בכל אחד מהרכיבים- הסנסורים, הסימולטור, מנוע החיפוש ועוד.
* הבנת עולם החיישנים: ניתן לשאול מהי מטרת חיישן מסוים, באילו יחידות הוא פועל, היכן הוא ממוקם וכדומה.
* שליפת ערך בזמן אמת: מסוגל להחזיר ערכים נוכחיים מתוך חיישני המערכת (לדוגמה: "מה הטמפרטורה בחוץ עכשיו?").
* שאלות כלליות: תומך גם בשאלות פשוטות (לדוגמה: "מה השעה עכשיו?") ושאלות טכניות בסיסיות.

1. **מסך חיפוש טכני (MQTT Search Engine):**

**תמונה שמכילה טקסט, צילום מסך, תוכנה, דף אינטרנט

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**

מנוע חיפוש פנימי, המבוסס על אינדוקס של מסמכים מאתר MQTT.org:

* מתבצעת יצירת אינדקס מקומי עם Stemming לשאילתות
* המשתמש מזין מילת מפתח או ביטוי חיפוש
* המערכת מציגה קישורים רלוונטיים למסמכים תואמים ישירות בתוך הדשבורד

1. **מסך סטטיסטיקות (Statistics Panel):**

**תמונה שמכילה טקסט, צילום מסך, תרשים, מספר

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**

מסך זה מאפשר לנתח ולחקור נתונים היסטוריים שנאספו מהחיישנים השונים במערכת.

הנתונים נאספים בכל דקה ומאוחסנים בענן לצורכי תיעוד והשוואה.

המערכת מאפשרת בחירת תאריך ושעת התחלה מתוך רשימה של ימים זמינים.

מטרת המסך היא לאפשר למשתמשים להפיק תובנות מתקדמות על תפקוד סביבת הייצור כמו זיהוי חריגות, מגמות ושינויים לאורך זמן.

1. **מסך סימולציית תקלות (Fault Simulator):**

**תמונה שמכילה טקסט, צילום מסך, מספר, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**

אתגר אינטראקטיבי המדמה תקלות חיישנים בסביבת פס הייצור. תהליך העבודה:

* בטבלה הראשונה מוצגות תקלות פעילות, ובטבלה השנייה- תקלות שטופלו. יש לבחור תקלה מהטבלה הראשונה.
* פרטי התקלה יופיעו בצד השמאלי של המסך תחת "FAULT DETECTED".
* בצד הימני יופיע "REPAIR CHALLENGE". יש ללחוץ Start Repair. משלב זה נמדד הזמן.
* המשתמש מבצע סימולציה של פעולות התיקון במעבדה. יש לסמן כל פעולה שהתבצעה בעזרת Checkbox.
* לאחר סיום- לוחצים Finish Repair. הניקוד מחושב לפי חומרת התקלה ומשך זמן התיקון.
* מוצגת הודעה עם סך הנקודות שהושגו. ניתן לבחור תקלה נוספת או לצאת.

1. **מסך אלפון (User Directory):**

**תמונה שמכילה טקסט, צילום מסך, מספר, מקביל

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**

רשימה של אנשי קשר מורשים במערכת, הכוללת:

* שם מלא
* תפקיד
* מספר טלפון

המסך משמש ליצירת קשר מהירה בין מהנדסים וסטודנטים במעבדה.

**מעברים בין מסכים:**

המעבר בין המסכים במערכת OptiLine מתבצע באופן מלא דרך תפריט הניווט הקבוע בצדו השמאלי של המסך. תפריט זה מופיע לאחר ההתחברות ונשאר זמין בכל זמן השימוש במערכת, גם תוך כדי ביצוע משימות או צפייה בנתונים.

כל לחיצה על כפתור בתפריט מחליפה את תוכן המסך המרכזי בלבד, מבלי לטעון מחדש את המערכת.

אין מעבר אוטומטי בין מסכים כתוצאה מפעולות פנימיות במערכת (למעט התחברות או התנתקות). כל מעבר הוא יוזמה של המשתמש דרך התפריט בלבד.

**הסבר על טעויות אפשריות:**

המערכת ידידותית ונוחה לתפעול, אך ייתכנו מצבים מסוימים שבהם המשתמש ייתקל בהודעות שגיאה או בהתנהגות לא צפויה. להלן פירוט של התרחישים העיקריים ודרכי ההתמודדות עימם:

1. **מצב תצוגה (Dark Mode):**

המערכת תומכת בשלב זה רק ב- Light Mode.

אם הממשק מוצג בצורה לקויה (למשל טקסט בהיר על רקע בהיר), יש לשנות את הגדרת ערכת הנושא (Theme) בדפדפן או בהגדרות ברירת המחדל של Gradio.

1. **פרטי משתמש לא נכונים:**

במקרה של הקלדת שם משתמש או סיסמה שגויים, תוצג הודעת שגיאה מתאימה. יש לוודא שהפרטים מוזנים בדיוק כפי שהוגדרו במערכת.

1. **טבלאות לא נפתחות בסימולטור תקלות:**

לעיתים, עקב מגבלות של ספריית Gradio, הטבלאות במסך סימולטור התקלות אינן נפתחות במלואן.

במקרה כזה יש ללחוץ עם העכבר על הטבלה עצמה- פעולה זו תגרום לפתיחתה המלאה.

1. **סימון משימות בתיקון תקלה:**

במהלך ביצוע אתגר תיקון בסימולטור חובה ללחוץ על Start Repair לפני שניתן לסמן את הצ'קבוקסים של שלבי התיקון. אם לא בוצעה לחיצה זו, הצ'קבוקסים לא יהיו פעילים.

1. **סיום תיקון- רק לאחר סימון כל השלבים:**

לא ניתן ללחוץ על Finish Repair לפני שכל הצ'קבוקסים סומנו.

המערכת בודקת שהמשתמש סימן את כל שלבי התיקון לפני שהיא מאפשרת לסיים את הטיפול ולחשב ניקוד.

**שקיפות אלגוריתמית:**

מערכת OptiLine פועלת תוך הקפדה על שקיפות אלגוריתמית מלאה- כלומר, כל פעולה, ניקוד, תצוגה או המלצה שניתנים למשתמש מבוססים על מנגנונים ברורים, ניתנים להסבר, ונחשפים למשתמש במלואם. אין שימוש בהחלטות אוטומטיות המבוססות על לוגיקה נסתרת.

מרכיבי השקיפות במערכת:

1. **מדריך משתמש מפורט:**

* המשתמש מקבל הסבר ברור על כל רכיב במערכת- מה מטרתו, כיצד משתמשים בו, ומה תפקידו בלוגיקת העבודה הכוללת.
* מוסבר כיצד צוברים ניקוד בסימולטור התקלות- כולל התנאים לקבלת ניקוד, היחס בין חומרת התקלה לבין משך התיקון, וכלל שלבי הפעולה הנדרשים.
* אין מנגנון אקראי או לא ידוע- כל תרחיש מוצג במפורש, וכל פעולה שהמשתמש נדרש לבצע מוסברת באופן שקוף.

1. **הצגת מידע גולמי:**

* בלוח הסנסורים, הערכים המוצגים הם המידע הגולמי ישירות מהחיישנים, כפי שהוא מתקבל בפרוטוקול MQTT.
* לא מתבצע עיבוד, סינון או הסקה כלשהי על הנתונים טרם הצגתם למשתמש.

1. **שקיפות מלאה בתהליך תיקון תקלה:**

* בסימולטור מוצגים כל שלבי הטיפול בתקלה, כולל הסבר על כל שלב והוראות ברורות לביצוע.
* ניקוד הסיום מבוסס אך ורק על פרמטרים גלויים: חומרת התקלה ומשך זמן התיקון.
* הדירוג אינו מושפע מפרמטרים נסתרים או חוקים פנימיים שאינם ידועים למשתמש.

1. **לוח ניקוד גלוי:**

* לוח הדירוג (Leaderboard) מוצג בזמן אמת ומכיל שם משתמש וניקוד בלבד.
* הלוח נבנה מתוך תוצאות הסימולציה ואינו מושפע מנתוני רקע או היסטוריה של המשתמש.
* כל משתמש יכול לראות את מיקומו ואת מיקומם של אחרים- מה שמחזק את תחושת ההוגנות.

1. **עוזר חכם מבוסס בינה מלאכותית:**

* הצ'אטבוט משתמש במודל שפה מבוסס Google Gemini, ומספק מענה לשאלות תפעוליות וטכניות.
* המערכת אינה אוספת את שאלות המשתמש לצורכי זיהוי אישי, ואין שמירת היסטוריה פרטית של השיח.
* העוזר לא מתערב בניווט בין מסכים או מקבל החלטות בשמם של המשתמשים- אלא מסביר בלבד.

**\*\*\* מצורף סרטון שיווקי- OptiLine Promo Video EN-CC.mp4 \*\*\***