# מבוא למחשוב ענן - סמסטר חורף התשפו

**תרגיל בית 3** -– **עבודה בצוותי העבודה**

מועד הגשה: 18.1.26

המשימה בתרגיל זה: סיום בניית המערכת, - איטרצית פיתוח מס 2

|  |  |
| --- | --- |
| https://colab.research.google.com/drive/1tZA0aSP3NEwdfezaLHjrsgv1oREVA5Wk?usp=sharing | **קישור למחברת ההגשה:** |
| https://github.com/AhmadTawil1/PlantSense-AI | **קישור לתיקיית גיט:** |

שימו לב: למטלה זו שלושה חלקים

**חלק ראשון.** 10) נקודות.(

עליכם להגדיר את בעלי התפקידים לאיטרציה זו )יש להחליף מתרגיל בית (2 יש לרשום מי מהנדס.ת המערכת, האחראי.ת בתרגיל זה.

|  |  |  |  |
| --- | --- | --- | --- |
| **Acceptance**  בדיקת **test –**  קבלה | **משימות שהושלמו** | **משימות שהוקצו** | **שם חבר הצוות** |
| • המערכת נשארת רספונסיבית (לא "קופאת") בזמן טעינת נתונים.    • המעבר בין ה-Services חלק וללא שגיאות קריסה. | • נבנתה מחלקת MicroserviceBase לניהול תהליכי רקע.  • הופרדו הלוגיקה הכבדה (AI/Network) מהממשק הגרפי.  • בוצע איחוד כל הקוד למחברת אחת יציבה. | • הגדרת דרישות וארכיטקטורת RAG  • **תכנון ומימוש ארכיטקטורת Microservices**  • **ניהול תהליכונים (Threading) למניעת חסימות**  • **אינטגרציה של מודל Gemini 2.0 Flash** | אחמד טויל |
| • בביצוע אנליזה או בדיקת חיישן, המשתמש מקבל הודעת תוספת XP.    • תגים (Badges) נפתחים אוטומטית בעמידה ביעדים מוגדרים. | • מומשה מחלקת User המנהלת את התקדמות החקלאי.    • נוצר מסך **Rewards** המציג הישגים ויזואלית.    • שולבה לוגיקה המעניקה נקודות על כל פעולה במערכת. | • פיתוח לוגיקת האנליזה והפקת דוחות    • **פיתוח מודל משחוק (Gamification)**    • **בניית מנגנון צבירת XP ודרגות משתמש** | **סירין פאהום**    (Analysis & Logic) |
| • חריגה בערכי החיישן מייצרת מיידית התראה אדומה (Critical).    • התראות שטופלו מסומנות כסגורות ונעלמות מהרשימה הפעילה. | • נבנה אלגוריתם המזהה חריגות (יובש/חום קיצוני) בזמן אמת.    • פותח מסך **Alerts** עם יכולת סינון לפי חומרה.    • מומש מנגנון Resolve לסגירת טיפול בהתראה. | • תמיכה בעיבוד קבצי PDF ותוצאות    • **פיתוח מערכת התראות חכמה (Alerts)**    • **הגדרת ספי חריגה (Thresholds) לחיישנים** | **איה חרמה**    (Analysis & Logic) |
| • הממשק נראה אחיד, מקצועי ומזמין (צבעים, פונטים, כפתורים).    • כל האלמנטים יושבים בצורה רספונסיבית ומסודרת במסך. | • הוחלף עיצוב ברירת המחדל בעיצוב "שטוח" ונקי.    • נבנו רכיבי "Cards" למידע וכרטיסיות צמח מעוצבות.    • שופרה הנגישות והניווט בתצורת SPA (Single Page). | • פיתוח מסכי תצוגת נתונים    • **עיצוב ממשק משתמש מתקדם (UI/UX)**    • **הטמעת Global CSS למראה מודרני** | **אדם תחאוק**    (Frontend / UI) |
| • הגרפים מתעדכנים אוטומטית עם כל דגימה חדשה.    • היסטוריית החיישנים מוצגת בבירור ומאפשרת זיהוי מגמות. | • שולבה ספריית Matplotlib ליצירת גרפים דינמיים.    • נבנה מנגנון המרת גרפים לתמונות (Base64) לתצוגה מהירה.    • בוצעה אופטימיזציה לשליפת נתונים מ-Firebase. | • דגימת נתונים ולוגיקת חיישנים    • **ויזואליזציה מתקדמת (גרפים בזמן אמת)**    • **אופטימיזציית הצגת היסטוריית נתונים** | **ואל סויד**    (Backend / Sensors) |

***חלק שני: בניית המערכת*** 70) נקודות(עליכם לסיים את בניית המערכת.

בפרט, יש לוודא קיום של כלל האלמנטים הבאים:

נקודות( :

35)

.1 כל הפיצ'רים שהתחלתם לממש בתרגיל בית 2

- יש לוודא שכל המסכים תקינים ועובדים כראוי, **בקולאב**, ולעבוד עם firebase לצורך שמירת

האינדקס ונתוני החיישנים.

**המימוש בקוד:** המערכת מכילה מחלקת FirebaseREST המנהלת את כל התקשורת מול מסד הנתונים.

**איפה זה קורה:**

* **שמירת נתוני חיישנים:** הפונקציה save\_sensor\_data שומרת את נתוני הלחות, הטמפרטורה והחיישנים ישירות ל-Firebase תחת המשתמש.
* **שמירת אינדקס:** הפונקציה save\_rag\_index שומרת את מבנה הנתונים של מנוע החיפוש (Inverted Index) ל-Firebase כדי שהמידע יישמר.

- על כל המסכים להופיע באפליקציה יחידה, תא בודד. אין לשים פיצ'רים שונים בתאים שונים.

**כל המסכים מופיעים באפליקציה יחידה (תא בודד):**

**המימוש בקוד:** המערכת בנויה כ-Single Page Application (SPA) בתוך המחברת.

**איפה זה קורה:** מחלקת ה-App הראשית (בסוף הקוד) ומחלקת ה-Navigation מנהלות את כל המעברים. הניווט מתבצע על ידי ניקוי ה -content\_area וטעינה מחדש של הפונקציה הרלוונטית (render\_home, render\_plants, וכו') באותו מקום פיזי במסך, מבלי לעבור לתאים אחרים.

- יש לממש מנוע חיפוש מבוסס RAG המבוסס על מאמרים הרלוונטיים לאפליקציה שלכם.

מנוע זה צריך להשתמש ב- LLM חיצוני אשר יחזיר תשובות אמיתיות המבוססות על המאמרים.

**מנוע חיפוש מבוסס RAG עם LLM חיצוני ומאמרים רלוונטיים:**

**המימוש בקוד:** קיימת מחלקה שלמה בשם PlantRAGSystem.

**RAG ואינדקס:** המערכת בונה "Inverted Index" מתוך רשימת מאמרים אקדמיים בנושאי חקלאות ובינה מלאכותית (SAMPLE\_PAPERS) ומתוך קובצי PDF שהמשתמש מעלה.

**LLM חיצוני:** המערכת משתמשת במודל **Gemini 2.0 Flash** דרך הספרייה langchain\_google\_genai) כדי לנסח את התשובה הסופית על בסיס המאמרים שנמצאו.

- אין להשאיר קישורים או כפתורים למסכים שלא קיימים.

- יש לממש צטבוט שישולב במערכת, מבוסס AI )כפי שלמדתם בתרגול ,(9 העונה בצורה אינטליגנטית לשאלות המשתמש.

**צ'אטבוט מבוסס AI המשתלב במערכת:**

* **המימוש בקוד:** מסך ה-Chat (render\_chat).
* **איפה זה קורה:** הצ'אט אינו סתם "תיבת טקסט", אלא מחובר למנוע ה-RAG. הוא יודע לשמור היסטוריית שיחה (save\_chat\_history ב-Firebase), להבין הקשר ולענות בצורה אינטליגנטית על שאלות הקשורות לצמחים ולמחלות, תוך שימוש במודל השפה של גוגל.

- יש לוודא שניתן להריץ את הפרויקט מהמחברת בלבד , ללא כל צורך בהטענה של קבצים נוספים

או יציאה למערכת חיצונית )כגון .(FLASK ניתן לעשות שימוש ב ,gradio יש לוודא כי האתר עולה ופועל כיאות.

.2 שימוש ב microservices- )תרגול -(8 עליכם לעשות שימוש לפחות בשניים. פרטו באילו שירותים עשיתם שימוש, ומה היתרון בשימוש בשירותים אלו. 10) נקודות(

**השירותים (Microservices) בהם עשינו שימוש:**

**1. שירות עדכון חיישנים (SensorUpdateMicroservice)**

**תפקיד השירות:** שירות זה אחראי באופן בלעדי על התקשורת מול העולם החיצוני (IoT). הוא מתשאל API חיצוני (server-cloud-v645.onrender.com) כדי למשוך נתוני זמן-אמת (לחות, טמפרטורה, תאורה) עבור הצמחים השונים. הוא מעבד את המידע הגולמי, משווה אותו לטווחי הנורמה של כל צמח, ומחזיר סטטוס "תקין" או "אזהרה".

**היתרון בשימוש:**

**מניעת חסימת הממשק (Non-blocking UI):** קריאות רשת (HTTP Requests) לשרת חיצוני עלולות להיות איטיות או להיכשל. על ידי בידוד התהליך ל-Microservice שרץ ברקע (Thread נפרד), ממשק המשתמש נשאר רספונסיבי ומהיר גם אם השרת החיצוני מתעכב.

**הפרדת תלויות:** אם ה-API החיצוני משתנה או נופל, רק השירות הזה מושפע, ושאר המערכת ממשיכה לעבוד עם נתונים במטמון (Cache).

**2. שירות ניתוח מחלות AI (AnalysisMicroservice)**

**תפקיד השירות:** זהו שירות המבצע את המשימות "הכבדות" של המערכת. הוא טוען ומריץ שני מודלים שונים: מודל ראייה ממוחשבת (MobileNetV2) לזיהוי המחלה בתמונה, ומודל שפה (Gemini LLM) להפקת הסבר מילולי והמלצות לטיפול.

**היתרון בשימוש:**

**ניהול משאבים וביצועים:** טעינת מודלים של Deep Learning לזיכרון והרצת Inference הן פעולות כבדות הדורשות זמן מעבד/GPU. ביצוע הפעולות הללו ב-Microservice נפרד מונע את "קפיאת" המסך בזמן שהמשתמש ממתין לתוצאות.

**Lazy Loading:**השירות טוען את המודלים הכבדים לזיכרון רק בפעם הראשונה שנעשתה פנייה אליו (\_load\_models), מה שמאפשר למערכת לעלות מהר בהתחלה ולחסוך משאבים אם המשתמש לא משתמש בפיצ'ר הניתוח.

.3 מהם ה KPI- הרלוונטיים בפרויקט שלכם? הסבירו. 10) נקודות(

**ה-KPIs (מדדי הביצוע) המרכזיים בפרויקט:**

1. **מדד בריאות המערכת החקלאית (Average Plant Health Score)**

המטרה העליונה של המערכת היא שמירה על צמחים חיים. מדד זה בודק את ממוצע ה-health\_score של כל הצמחים הרשומים במערכת.

היעד הוא שמירה על ציון בריאות ממוצע של **מעל 80%** ציון ,"Healthy". בקוד מוגדר כי ציון מתחת ל-70% מקפיץ התראת Warning וציון מתחת ל-50% מקפיץ Critical.

1. **מדד זמן תגובת מסך (Screen Response Time)**

חווית המשתמש (UX) תלויה במעבר חלק ומהיר בין מסכים. מדד זה בודק את הזמן מרגע הלחיצה על כפתור ניווט ועד להופעת התוכן, תוך הסתמכות על מנגנון ה-Caching שבנינו.

היעד הוא זמן תגובה של פחות מ-1 שניה להצגת נתונים ברוב האינטראקציות. הדבר מתאפשר בזכות מנגנון ה-Cache שהוגדר בקוד לפרק זמן של 5 דקות (\_CACHE\_DURATION), המונע המתנה מיותרת לשרת החיצוני במעבר בין מסכים.

1. **מדד איכות ואמינות המודל (AI Model Confidence)**

רמת הוודאות של המודל (MobileNetV2) בזיהוי המחלה. מדד זה קריטי כדי למנוע הטעיית חקלאים עם אבחנות שגויות.

המערכת שואפת לממוצע רמת ביטחון (Confidence) של **מעל 85%**. הקוד מחשב את ה-confidence באחוזים (0-100), כאשר תוצאות נמוכות יגררו המלצה למשתמש לצלם שוב את העלה בתנאי תאורה טובים יותר.

.4 מידע עתק: השתמשו במודלים שהוצגו בשבוע 9 ובצעו ניתוח על המידע שהגיע מהחיישנים ושמרתם ב .DB הציגו בוורד גרף נתונים של חיתוך לבחירתכם 5) נקודות.(

בפרויקט זה, ביצענו ניתוח של נתוני העתק (Big Data) המגיעים מחיישני ה-IoT באמצעות מודלים של ויזואליזציה וניתוח מגמות (Trend Analysis). הנתונים נשמרים במסד הנתונים (Firebase) ונשלפים לצורך ניתוח היסטורי.

* **המודל:** שימוש ב-Time-Series Analysis כדי לזהות חריגות ומגמות בטמפרטורה, לחות הקרקע ולחות האוויר.
* **המימוש:** המערכת מושכת את 50 הקריאות האחרונות (limit=50) עבור כל צמח, ומייצרת בזמן אמת גרפים המאפשרים למשתמש לראות את יציבות הסביבה של הצמח לאורך זמן.

**גרף נתונים (חיתוך טמפרטורה לאורך זמן):**

A graph of different types of data

AI-generated content may be incorrect.

**הסבר לגרף:** הגרף מציג את שינויים לאורך ציר הזמן (קריאות החיישן). ניתוח זה מאפשר לזהות "גלי חום" או "מכות קור" שעלולים לפגוע בצמח, גם אם הטמפרטורה הנוכחית תקינה. זוהי דוגמה לשימוש במידע שנצבר (Data) כדי להפיק תובנות (Insights) שאינן זמינות במבט רגעי בודד.

.5 אתגרים שעלו במהלך העבודה , וכיצד התמודדתם איתם )לדוגמא- DB גדול מדי, יצירת ויזואליזציה, מבנה הקוד וכו.( 5) נקודות.(

במהלך הפיתוח נתקלנו במספר אתגרים טכניים ועיצוביים, שנבעו מהאילוץ לאחד מערכת מורכבת לתוך סביבת ריצה של מחברת (Notebook):

**1. אתגר חסימת הממשק (UI Freezing) בעת קריאות רשת:**

בכל פעם שהמערכת ניסתה למשוך נתונים מה-API של החיישנים או לשלוח תמונה לניתוח ב-Gemini, הממשק הגרפי "קפא" והפך ללא-רספונסיבי עד סיום הפעולה. זאת מכיוון שבסביבת Python רגילה הקוד רץ בצורה סדרתית (Synchronous).

**הפתרון:** יישמנו ארכיטקטורת **Microservices-in-Notebook**. בנינו מחלקת בסיס MicroserviceBase המשתמשת ב-threading וב-queue של Python. כל שירות כבד (חיישנים, AI) רץ כעת ב-Thread נפרד ברקע ("Daemon Thread"), מה שמאפשר לממשק המשתמש להישאר מהיר ופעיל תמיד, בעוד הנתונים מתעדכנים אסינכרונית.

**2. אתגר עיצוב ממשק מודרני בתוך Jupyter (Custom CSS & Styling):**

**האתגר:** ספריית ipywidgets מספקת ברירת מחדל ויזואלית בסיסית ו"מדעית" מאוד, שאינה מתאימה לאפליקציית משתמש (Consumer App). האתגר היה ליצור מראה מודרני, נקי ומותאם לנושא החקלאות (צבעי ירוק/אדמה) בתוך המגבלות של המחברת.

**הפתרון:** הזרקנו בלוק GLOBAL\_CSS מקיף בתחילת הריצה שדורס את סגנונות ברירת המחדל של Jupyter. הגדרנו פלטת צבעים אחידה (משתני CSS כגון --primary: #2ecc71), עיצבנו כפתורים עגולים ומודרניים, והשתמשנו ב-widgets.HTML כדי לבנות רכיבי UI מורכבים כמו "Cards" ו-"Badges" שלא קיימים בספרייה המקורית.

**3. אתגר ניהול מבנה הקוד (Code Structure & Complexity):**

**האתגר:** הדרישה לאחד את כל הפיצ'רים לתא בודד יצרה קובץ קוד ענק שקשה לתחזוקה ולניווט. היה חשש מ"קוד ספגטי" שבו לוגיקה של מסד נתונים מתערבבת עם לוגיקה של תצוגה.

**הפתרון:** אימצנו עקרונות **OOP** (תכנות מונחה עצמים) ו-**Separation of Concerns**. חילקנו את הקוד למחלקות ברורות לפי תפקיד: FirebaseREST לנתונים, PlantRAGSystem ללוגיקה, ופונקציות render\_page נפרדות לכל מסך. מחלקת App ראשית מנהלת את הניווט ביניהן בתבנית Single Page Application, מה שאפשר עבודה מסודרת ונקייה.

.6 יש להוסיף ממשק כלשהו של משחוק כרצונכם )אלמנט בודד כגון צבירת נקודות, תגים, שלבים וכו.( 5) נקודות.(

שילבנו במערכת מודול משחוק מלא שמטרתו לעודד את המשתמשים לתחזק את הצמחים ולנטר אותם באופן יומיומי. הממשק כולל שלושה אלמנטים מרכזיים:

1. **מערכת דרגות ו-XP (Experience Points):**
   * כל פעולה במערכת (הוספת צמח, בדיקת חיישנים, ניתוח AI) מעניקה למשתמש נקודות ניסיון (XP).
   * המשתמש מתקדם בין 5 דרגות אבולוציוניות של חקלאי: מ-**"Seedling"** (שתיל) ועד **"Forest Guardian"** (שומר היער), כאשר כל דרגה דורשת סף XP גבוה יותר.
2. **תגים והישגים (Badges & Achievements):**
   * יצרנו גלריית תגים ויזואלית שנפתחת אוטומטית בעמידה ביעדים.
   * לדוגמה: תג **"Green Thumb"** לצבירת רצף פעילות, תג **"Sensor Pro"** לביצוע 50 בדיקות חיישנים, ותג **"AI Enthusiast"** לשימוש במודלי הניתוח.
3. **משימות יומיות (Daily Tasks):**
   * רשימת משימות מתחלפת ("Check Sensors", "Water Plants") שמתאפסת כל 24 שעות. השלמת כל המשימות מעניקה בונוס XP משמעותי ותורמת למדד ה- **Streak** (רצף ימים), מה שמעודד כניסה יומית לאפליקציה.

**מימוש טכני:** כל נתוני ההתקדמות נשמרים ב-Firebase תחת האובייקט rewards של המשתמש, מה שמאפשר שמירת ההישגים גם במעבר בין סשנים שונים.

A screenshot of a computer

AI-generated content may be incorrect.

***חלק שלישי : סגירת הפרויקט )\_ 20 נקודות(***

הכינו מסמכים המתארים את הפרויקט שלכם:

.1 יש לבנות תיק למתכנת הכולל את שמות כל הקבצים המרכזיים, פונקציות מרכזיות, קטעי קוד/תבניות עיצוב מעניינים שהשתמשתם בהם.

**1. ארכיטקטורת המערכת ומבנה הקוד**

למרות שהמערכת רצה בתוך תא מחברת (Notebook Cell) בודד, הקוד בנוי בארכיטקטורה מודולרית המבוססת על עקרונות  **OOP (Object Oriented Programming)**. המערכת מחולקת לשכבות לוגיות ברורות:

* **Logic Layer (Business Logic & AI):** מחלקות המנהלות את הלוגיקה העסקית, עיבוד התמונה ומודלי השפה.
* **Data Access Layer (DAL):** מחלקות המנהלות את התקשורת מול Firebase.
* **Presentation Layer (UI):** פונקציות רינדור המשתמשות ב-ipywidgets.
* **Service Layer:** שירותי רקע א-סינכרוניים (Microservices).

**רכיבים מרכזיים (Key Components / Classes):**

| **שם המחלקה** | **תיאור ותפקיד במערכת** |
| --- | --- |
| App | **Main Controller**. מחלקת ה-Entry Point המנהלת את מחזור חיי האפליקציה, את ה-State הגלובלי (CURRENT\_USER) ואת ניתוב המסכים (Routing). |
| MicroserviceBase | **Base Class**. מחלקה אבסטרקטית המממשת תבנית **Producer-Consumer** עם Threading ו-Queue. מאפשרת הרצת שירותים כבדים ברקע מבלי לחסום את ה-UI. |
| AnalysisMicroservice | **AI Service**. יורש מ-MicroserviceBase. מנהל Pipeline היברידי: MobileNetV2 לזיהוי ויזואלי + Gemini LLM להסבר מילולי. |
| PlantRAGSystem | **Logic**. מנוע חיפוש סמנטי. מממש Inverted Index, טעינת מסמכים ואינטגרציה עם LLM למתן תשובות מבוססות מחקר (RAG). |
| FirebaseREST | **DAL**. עוטף את קריאות ה-REST ל-Firebase ומנתק את הלוגיקה העסקית מהתקשורת הישירה עם ה-DB. |

**פונקציות מרכזיות (Core Functions)**

להלן הפונקציות הקריטיות המניעות את הליבה הטכנולוגית:

1. **process(self, data) בתוך AnalysisMicroservice**
   * **הלוגיקה:** פונקציה זו מנהלת את שרשרת הניתוח המורכבת. היא מקבלת Bytes של תמונה --> טוענת מודלים בשיטת Lazy Loading -->מבצעת Inference ב-PyTorch לקבלת סיווג ראשוני --> בונה Prompt דינמי ב-LangChain ושולחת ל-Gemini לקבלת ניתוח מובנה (JSON).
2. **query(self, question, ...) בתוך PlantRAGSystem**
   * **הלוגיקה:** המוח של הצ'אטבוט. הפונקציה מבצעת Preprocessing לשאילתה, סורקת את האינדקס ההפוך, מדרגת תוצאות לפי רלוונטיות, ובונה את ה-Context לחלון ההקשר של ה-LLM כדי למנוע הזיות (Hallucinations).
3. **create\_history\_graph(history\_data) בתוך render\_sensors**
   * **הלוגיקה:** פונקציית ויזואליזציה הממירה נתונים גולמיים לגרף ויזואלי בתוך ה-Browser. היא משתמשת ב-matplotlib ליצירת הגרף בזיכרון, ממירה אותו ל- Base64 String, ומזריקה אותו דינמית ל-HTML, מה שמאפשר עדכון גרפים בזמן אמת ללא רענון עמוד.

**ארכיטקטורת AI ושימוש ב-LangChain**

המערכת מדגימה שימוש מתקדם בספריית **LangChain** כשכבת ניצוח (Orchestration) המחברת בין המודלים השונים.

**א. תהליך הניתוח ההיברידי (Hybrid AI Pipeline)**

במקום להסתמך רק על סיווג תמונה (שיכול להיות לקוני, כמו "Late\_blight"), המערכת מבצעת העשרה סמנטית:

1. **Vision:** המודל MobileNetV2 מזהה את המחלה טכנית.
2. **Language:** התוצאה מוזרקת ל-LangChain, שמשתמש ב-Gemini 2.0 Flash כדי לתרגם את המונח הטכני להסבר ידידותי, להעריך חומרה, ולהפיק המלצות פעולה ("השקה פחות", "הוסף דשן").

**ב. הנדסת פרומפטים (Prompt Engineering)**

נעשה שימוש בטכניקות מתקדמות של Prompting:

* **Role Prompting:** הגדרת המודל כמומחה ("You are an expert plant pathologist").
* **Structured Output:** שימוש ב-PydanticOutputParser כדי לכפות על המודל להחזיר תשובה במבנה JSON תקין ולא בטקסט חופשי. זה קריטי לצורך הצגת הנתונים ב-UI (למשל, צביעת תגית חומרה באדום או ירוק בהתאם לשדה severity ב-JSON).

**דוגמת קוד מהמערכת (הגדרת ה-Prompt):**

# מתוך מחלקת AnalysisMicroservice

self.prompt = ChatPromptTemplate.from\_messages([

("system", "You are an expert plant pathologist..."),

("human", """Analyze this plant disease diagnosis:

Plant Name: {plant\_name}

Detected Condition: {disease\_name}

Confidence: {confidence}%

{format\_instructions} # הזרקה אוטומטית של הוראות הפרסור

Rules:

- If healthy, severity should be "Low".

- Provide 3-5 specific recommendations.""")

])

**תבניות עיצוב וקטעי קוד מעניינים (Design Patterns)**

**א. מימוש Microservices בתוך Notebook**

כדי למנוע חסימה של ממשק המשתמש (Blocking UI) בעת קריאות רשת כבדות, מימשנו תבנית **Async Worker** באמצעות Threads.

**קטע קוד (MicroserviceBase):**

Python

class MicroserviceBase:

def \_\_init\_\_(self, name: str):

self.request\_queue = queue.Queue() # Thread-safe queue

def start(self):

# הרצת ה-Worker ב-Thread נפרד (Daemon)

self.worker\_thread = threading.Thread(target=self.\_worker, daemon=True)

self.worker\_thread.start()

def request(self, data, timeout=30.0):

# הכנסת בקשה לתור והמתנה מסונכרנת לתוצאה (מבלי לעצור את ה-Main Loop של ה-UI)

request\_id = f"{self.name}\_{time.time()}"

self.request\_queue.put((request\_id, data))

**ב. Lazy Loading (טעינה עצלה)**

אופטימיזציה לזמן עלייה: מודלי ה-AI נטענים לזיכרון ה-RAM רק בפעם הראשונה שהמשתמש לוחץ על כפתור "Analyze", ולא בעת הפעלת האפליקציה.

**קטע קוד:**

def \_load\_models(self):

if self.model is None: # בדיקה האם המודל כבר נטען

print(" Loading MobileNetV2 Plant Disease Model...")

self.processor = AutoImageProcessor.from\_pretrained(...)

self.model = AutoModelForImageClassification.from\_pretrained(...)

.2 יש לבנות תיק למשתמש , הכולל הסבר כללי על המערכת , פירוט מסכים, מעברים בין מסכים והסבר על טעויות אפשרויות.

**תיק למשתמש - מדריך הפעלה למערכת PlantSense-AI**

**1. מבוא והסבר כללי**

**PlantSense-AI** היא פלטפורמה חכמה לניהול חקלאי המשלבת בינה מלאכותית (AI) וחיישנים (IoT). המערכת מאפשרת לך ליצור "תאום דיגיטלי" של הגינה שלך, לנטר מדדים בזמן אמת, לקבל אבחון מחלות אוטומטי, ולהתייעץ עם בוט מומחה. המערכת פועלת כולה מתוך הדפדפן (Google Colab) ואינה דורשת התקנת חומרה מיוחדת במחשב.

**2. התחברות ורישום (Getting Started)**

לפני השימוש במערכת, יש ליצור חשבון אישי. המערכת שומרת את הנתונים שלך בצורה מאובטחת בענן (Firebase).

* **הרשמה:** לחץ על "Create New Account", בחר שם משתמש וסיסמה.
* **התחברות:** הזן את פרטיך ולחץ על "Login".

A screenshot of a computer

AI-generated content may be incorrect.

**3. פירוט המסכים (Screen Guide)**

**א. מסך הבית (Home Dashboard)**

זהו מרכז הבקרה שלך. המסך מרכז את כל המידע הקריטי במקום אחד.

* **סטטיסטיקות מהירות:** כמות הצמחים, התראות פעילות, ורמת המשתמש שלך (Level).
* **פעולות מהירות (Quick Actions):** כפתורי קיצור להוספת צמח, בדיקת חיישנים או התחלת צ'אט.
* **צמחים אחרונים:** תצוגה מקוצרת של 3 הצמחים האחרונים שעודכנו.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**ב. ניהול הגינה (Plants Page)**

כאן תוכלו לראות את ה"חממה הדיגיטלית" שלכם.

* **הוספת צמח:** הזינו שם ובחרו את סוג הגידול (למשל: Tomato, Corn) מתוך הרשימה.
* **כרטיסיות צמח:** לכל צמח יש כרטיסייה המציגה את ציון הבריאות שלו (0-100%), סטטוס מחלה אחרון ומדדי חיישנים.

A screenshot of a computer

AI-generated content may be incorrect.

**ג. ניטור חיישנים (Sensors Page)**

מסך זה מציג נתונים בזמן אמת המגיעים מהחיישנים בשטח.

* **שעונים (Gauges):** מציגים ויזואלית את הלחות, הטמפרטורה והלחות באוויר. הצבע משתנה (ירוק/צהוב) בהתאם לתקינות.
* **גרפים היסטוריים:** מאפשרים לראות מגמות לאורך זמן (למשל: האם הטמפרטורה צנחה בלילה?).

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a graph

AI-generated content may be incorrect.

**ד. מעבדת ה-AI (Analysis Page)**

זהו הלב החכם של המערכת.

1. בחרו צמח מהרשימה.
2. העלו תמונה של העלה החשוד.
3. לחצו על **Analyze Image**. המערכת תציג אבחון רפואי הכולל: שם המחלה, רמת ביטחון (%), הסבר מילולי והמלצות לטיפול.

* **הפקת דוח:** ניתן ללחוץ על "Download PDF Report" לקבלת מסמך רשמי.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

**ה. העוזר האישי (Chat Page)**

צ'אטבוט חכם שיודע לענות על שאלות חקלאיות.

* **העלאת מסמכים:** ניתן להעלות קובץ PDF (למשל דוח מעבדה) והבוט "ילמד" אותו ויענה על שאלות מתוכו.
* **שקיפות:** הבוט מציג את המקורות עליהם הוא מתבסס ("Retrieved From").

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**ו. אזור המשחוק (Rewards Page)**

כדי לעודד התמדה, המערכת מתגמלת אתכם על פעילות.

* **סרגל XP:** מציג כמה נקודות חסרות לדרגה הבאה (למשל: מ-Seedling ל-Sprout).
* **תגים (Badges):** גלריה של הישגים שנפתחו (למשל "Green Thumb" על רצף פעילות).

A screenshot of a computer

AI-generated content may be incorrect.

**ז. מרכז ההתראות (Alerts Page)**

מסך זה הוא "חדר המצב" של הגינה שלכם, המרכז את כל האירועים החריגים הדורשים טיפול מיידי.

* **סינון חכם:** בראש המסך קיימים פילטרים המאפשרים למיין התראות לפי צמח ספציפי, סוג הבעיה (חיישן/מחלה), או חומרת המצב.
* **צבעי חומרה:** כרטיסיות **אדומות** מסמנות מצב קריטי (Critical) הדורש פעולה מיידית (למשל: יובש קיצוני), בעוד כרטיסיות **צהובות** מסמנות אזהרה (Warning).
* **פעולות:**
  + **Mark Read:** סימון שראיתם את ההתראה אך טרם טיפלתם בה.
  + **Resolve:** אישור שהבעיה טופלה (למשל: השקיתם את הצמח), מה שיעביר את ההתראה לארכיון.

A screenshot of a computer

AI-generated content may be incorrect.

**4. מעברים וזרימת עבודה (Navigation Flow)**

הניווט במערכת תוכנן להיות שטוח ומהיר:

1. **סרגל ניווט עליון:** נמצא תמיד בראש המסך ומאפשר מעבר מיידי בין כל המודולים (Home, Plants, Sensors וכו') ללא צורך בכפתור "חזור".
2. **זרימה חכמה:**
   * זיהיתם בעיה ב**מסך החיישנים**? עברו ל**מסך האנליזה** לבדיקת תמונה.
   * קיבלתם תוצאה לא ברורה ב**אנליזה**? עברו ל**צ'אט** לשאול את הבוט מה לעשות.

**5. פתרון תקלות נפוצות (Troubleshooting)**

| **הודעת שגיאה / תופעה** | **סיבה אפשרית** | **פתרון מומלץ** |
| --- | --- | --- |
| **"Analysis Timeout"** | בשימוש ראשון, המערכת טוענת מודלים כבדים לזיכרון. | המתינו כדקה ונסו שוב. הפעמים הבאות יהיו מהירות הרבה יותר. |
| **"No leaf detected"** | המודל לא זיהה עלה בתמונה שהעליתם. | צלמו את העלה מקרוב, על רקע ניטרלי ובתאורה טובה. |
| **גרפים ריקים (Sensors)** | אין עדיין היסטוריית נתונים עבור הצמח הזה. | לחצו על "Refresh All Data" כדי למשוך נתונים חדשים מהחיישן. |
| **"Firebase not accessible"** | בעיית חיבור לאינטרנט או חסימת רשת. | ודאו חיבור לאינטרנט. הנתונים יישמרו זמנית בזיכרון המקומי עד לחידוש הקשר. |

.3 התייחסו בתיק המשתמש לשקיפות אלגוריתמית – כיצד הבהרתם למשתמשים את האלגוריתמים והנתונים הנאספים בקוד שלכם?

**שקיפות אלגוריתמית (Algorithmic Transparency)**

ב-PlantSense-AI, אנו מאמינים שהמשתמש צריך להבין כיצד המערכת מקבלת החלטות. לכן, הטמענו מספר מנגנונים החושפים את "מאחורי הקלעים" של האלגוריתמים:

**1. חשיפת מדד הביטחון (Confidence Score):** המערכת אינה מספקת תשובות בינאריות ("חולה/בריא"), אלא מציגה שקיפות מלאה לגבי רמת הוודאות של המודל.

* **איך רואים את זה?** במסך האנליזה, לצד האבחון, מופיע "בר" צבעוני המציג את אחוז הביטחון (למשל: 87%).
* **המשמעות:** אם הציון נמוך (מתחת ל-50%), המערכת תזהיר אתכם שייתכן והאבחון אינו מדויק ותמליץ לצלם שוב.

**2. הסבר החלטה (Explainable AI):** במקום לזרוק מונח רפואי לטיני (למשל "Septoria Leaf Spot"), המערכת משתמשת במודל שפה (Gemini) כדי להסביר **במילים פשוטות** מדוע התקבלה ההחלטה.

* **איך רואים את זה?** תחת כל אבחון מופיעה פסקה בשם **"Explanation"** המתארת את הסימפטומים שהמערכת זיהתה בתמונה.

**3. ציטוט מקורות בצ'אט (Source Attribution):** כאשר אתם שואלים את הבוט שאלה, הוא אינו ממציא תשובות. מנגנון ה-RAG (Retrieval-Augmented Generation) שלנו סורק מאמרים ודוחות PDF שהעליתם.

* **איך רואים את זה?** בסוף כל תשובה בצ'אט, מופיעה תיבת **" Retrieved From"** המפרטת בדיוק באילו מאמרים או קבצים המערכת השתמשה כדי לבנות את התשובה.

**4. נתונים גולמיים מול מעובדים:** אנו מציגים למשתמש את הנתונים הגולמיים (Raw Data) בגרפים, ולא רק את הסטטוס המעובד ("תקין/לא תקין").

* **איך רואים את זה?** במסך החיישנים, ניתן לראות את היסטוריית הטמפרטורה המדויקת בגרף, כך שתוכלו לוודא בעצמכם אם החלטת המערכת להוציא התראה הייתה מוצדקת.

.4 יש להכין סרטון קצר של 30-60 שניות, המתאר את השימוש במערכת. הסרטון משמש כ –

pitch elevator למערכת שלכם, כלומר יש לכלול בו הסבר מקצועי ועם זאת שיווקי , המדגיש

את האלמנטים המיוחדים של המערכת שבניתם. יש להגיש את הסרטון בפורמט .4mp



click twice

הוראות הגשה:

.1 יש להגיש את התרגיל בצוותים, בתיקיית ה GIT– שלכם, וכן בתיקייית התרגיל ב .moodle

# חובה לכלול קישור לתיקיית הגיט בקובץ במוודל.

ובו

.2 יש להגיש במוודל קובץ זיפ הכולל קובץ וורד ובו מענה לשאלות, וקישור ל- notebook

הקוד שלכם )יש לוודא שהקישור פומבי ונגיש.(

.3 כותרתו של הקובץ תהיה HW3\_TEAMNAME

.4 שימו לב כי כל העבודות חייבות להיות שונות זו מזו.אנו מריצות תוכנה לבדיקת עבודות זהות. עבודות שייראו דומות ייפסלו ויינתן עליהן ציון .0

בהצלחה!