**1. תיאור המערכת**

**1.1 מטרת הפרויקט**

המערכת שאני מפתחת נועדה להקל על מתכנתים בניהול, אופטימיזציה ותמיכה בקוד. זהו פלטפורם לניהול קבצי קוד, שבו המשתמשים יוכלו להעלות קבצים, לשייך להם תגיות, לאבחן בעיות בקוד ולבצע אופטימיזציה בצורה אוטומטית. המערכת מציעה פתרונות יעילים לשיפור ביצועים ומניעת שגיאות.  
הערך המוסף: **קלות השימוש**, **אוטומציה** ויכולת **שיתוף פעולה** לשיפור מהירות ודיוק בקוד.

תכונות עיקריות:

* העלאת קבצים לצורך אופטימיזציה ובדיקות קוד.
* אוטומטיזציה של תיקון שגיאות והצעות לשיפור.
* ניהול קבצים בענן.
* כלי תיוג לקבצים לשיפור הארגון.
* ממשק משתמש אינטואיטיבי.

**1.2 קהל היעד**

המערכת פונה למתכנתים מקצועיים ולחברות פיתוח המעוניינות לשדרג את תהליך ניהול הקוד, תיקול בעיות ואופטימיזציה. קהל היעד יכלול:

* **מתכנתים עצמאיים** שמחפשים כלים לניהול הקוד בצורה חכמה.
* **חברות טכנולוגיה** שרוצות לשפר את תהליך הפיתוח שלהן עם אוטומציה ובדיקות קוד בזמן אמת.
* **חברות שמחפשות פתרונות לניהול קוד בענן** עם יכולת שיתוף ושיתוף פעולה בין צוותים.

**1.3 פונקציונליות של המערכת**

המערכת תספק את הפונקציות העיקריות הבאות:

1. **העלאת קבצים**: המשתמשים יוכלו להעלות קבצים בשפות תכנות שונות, כמו JavaScript, Python ו-C#.
2. **בדיקות קוד**: המערכת תבצע אוטומטית סריקות קוד, תמצא בעיות ותחזיר הצעות לשיפור.
3. **אופטימיזציה של קוד**: המערכת תספק דרכים לשיפור ביצועים ותיקון בעיות אוטומטית.
4. **ניהול קבצים בענן**: ניהול קבצים בצורה מבוזרת בענן עם אפשרות לשיתוף בין משתמשים.
5. **תיוג קבצים**: המערכת תאפשר שיתוף ותיוג הקבצים בצורה מסודרת.
6. **היסטוריית תיקונים**: שמירת היסטוריה של תיקונים ואופטימיזציות לכל קובץ.

**1.4 בעיות שהמערכת פותרת**

המערכת פותרת את הבעיה של ניהול קבצים מבולגן ותחזוקה מסובכת של קוד בין פרויקטים שונים. היא מספקת פתרון כולל הכולל:

* **אופטימיזציה אוטומטית** לקוד כדי למנוע שגיאות נפוצות ולייעל ביצועים.
* **ניהול קבצים בענן** שמקל על שיתוף קוד בין צוותים.
* **שיפור איכות הקוד** בזמן אמת באמצעות תיקון בעיות שמזוהות במהלך העלאת הקבצים.

המערכת מבטיחה שהמשתמשים יוכלו להימנע מתקלות מיותרות ולבצע אופטימיזציה בקוד בצורה קלה ומהירה!

**1.5 טכנולוגיות בשימוש**

הפרויקט יפותח באמצעות הטכנולוגיות הבאות:

1. **Backend**: Node.js ו-Express.js לפיתוח ה-API שיבצע את כל העיבוד בצד השרת.
2. **Frontend**: React.js לבניית ממשק משתמש אינטראקטיבי.
3. **Database**: MongoDB לאחסון נתונים ומידע אודות הקבצים.
4. **Cloud Storage**: AWS S3 לאחסון הקבצים בפתרון ענן.
5. **AI Tools**: שימוש בטכנולוגיות של NLP ו-AI למתן הצעות שיפור לקוד ולביצוע Code Review.

**1.6 מתודולוגיית פיתוח (Agile)**

הפיתוח יתבצע במתודולוגיית Agile עם חלוקה לספרינטים. כל ספרינט יתמקד במאפיינים עיקריים שיביאו את המערכת לרמה גבוהה יותר של יכולת עבודה.

**ספרינט 1 - תשתית בסיסית**:

* תכנון מסד נתונים ב-MongoDB.
* פיתוח API בסיסי עם Express.js.
* הגדרת אחסון ענן ב-AWS S3.

**ספרינט 2 - ממשק משתמש**:

* בניית ממשק React.js.
* חיבור בין ה-Frontend ל-Backend.
* פיתוח יכולת העלאת קבצים.

**ספרינט 3 - אופטימיזציה ובדיקות קוד**:

* פיתוח כלי לזיהוי בעיות בקוד והצעות לשיפור.
* יצירת אלגוריתמים של אופטימיזציה באמצעות AI.

**ספרינט 4 - אבטחה ושיפורים**:

* הטמעת אמצעי אבטחה מתקדמים.
* שיפור ביצועים וייעול הקוד.

**ספרינט 5 - פריסה ותחזוקה**:

* פריסת המערכת בסביבת AWS.
* תחזוקה ושדרוגים.

**2. עיצוב ממשק משתמש ויזואלי**

**2.1 מסך כניסה/רישום**

**תיאור המסך**: המסך הראשון במערכת, שבו המשתמשים יכולים להתחבר או להירשם לשירות.

* **לחיצה על כפתור "הירשם"**: תוביל למסך רישום, שם המשתמש יזין את הפרטים האישיים.
* **לחיצה על "התחבר"**: תבצע אימות משתמש ותעביר למסך ניהול הקבצים.

**הסבר על זרימת המשתמש**:

1. המשתמש יראה שני כפתורים עיקריים: "התחבר" ו"הירשם".
2. כפתור "התחבר" יוביל למסך ההתחברות בו יש להכניס דוא"ל וסיסמה.
3. כפתור "הירשם" יוביל למסך רישום בו יוכל להזין פרטים אישיים וליצור חשבון חדש.

**2.2 מסך ניהול קבצים**

**תיאור המסך**: זהו המסך המרכזי בו המשתמשים יוכלו לנהל את הקבצים שלהם. המסך יציג את כל הקבצים שהועלו, עם אפשרויות כמו עריכה, מחיקה ותיוג.

* **רשימת קבצים**: הצגת קבצים שהמשתמש העלה עם כפתורי פעולה.
* **כפתור "העלאת קובץ"**: יאפשר למשתמש להעלות קבצים חדשים.
* **כפתור "הוספת תגית"**: יאפשר להוסיף תגית לכל קובץ לצורך ניהול וסינון.

**הסבר על זרימת המשתמש**:

1. המשתמש יראה רשימה של הקבצים שהועלו בעבר.
2. כפתור "העלאת קובץ" יפתח דיאלוג לבחירת קובץ מהמחשב.
3. לחיצה על כפתור "הוספת תגית" תאפשר למשתמש להוסיף תגית לכל קובץ.

**2.3 מסך אדמין**

**תיאור המסך**: מסך המיועד למנהלי מערכת שמציג סטטיסטיקות שימוש ומידע על הפעילות במערכת.

* **סטטיסטיקות פעילות**: מידע על מספר הקבצים שהועלו, מספר המשתמשים ועוד.
* **ניהול משתמשים**: אפשרות למחוק או לעדכן מידע על משתמשים.

**הסבר על זרימת המשתמש**:

1. המנהל יוכל לראות את הסטטיסטיקות של המערכת בצורה גרפית (כמו תרשימים או טבלאות).
2. כפתור "ניהול משתמשים" יוביל למסך בו ניתן לנהל את המשתמשים, למחוק או לעדכן אותם.

**2.4 פירוט תהליכים מרכזיים**

* **העלאת קובץ**:
  + לאחר לחיצה על כפתור "העלאת קובץ", המשתמש יבחר קובץ מהמחשב.
  + הקובץ יועלה למערכת ויתווסף לרשימת הקבצים האישית של המשתמש.
* **הוספת קטגוריה/תגית לקובץ**:
  + המשתמש ילחץ על כפתור "הוספת תגית".
  + לאחר מכן, הוא יבחר תגית מתוך רשימה או ייצור אחת חדשה.
  + התגית תתווסף לקובץ ותשפר את ארגון הקבצים.

**3.1 רשימת פונקציות המערכת**

**1. העלאת קובץ**

* **Route**:  
  POST /api/upload
* **פרמטרים**:  
  **Body**: { file: File, tags: string[] }  
  **Headers**: Authorization: Bearer <JWT\_TOKEN>
* **פלט**:  
  data: { fileId: string, url: string }  
  **קודי סטטוס**: 200 OK, 400 Bad Request (במקרה של קובץ לא תואם)
* **לוגיקה עסקית**:
  + הגבלת גודל קובץ ל-50MB.
  + הוספת חתימת זמן לקובץ.
  + שייך את הקובץ למשתמש המאמת את עצמו.

**2. ניהול קבצים (הצגת קבצים)**

* **Route**:  
  GET /api/files
* **פרמטרים**:  
  **Headers**: Authorization: Bearer <JWT\_TOKEN>
* **פלט**:  
  data: { files: File[] }  
  **קודי סטטוס**: 200 OK, 401 Unauthorized
* **לוגיקה עסקית**:  
  הצגת כל הקבצים שהועלו על ידי המשתמש המאמת את עצמו, כולל מידע על תגיות אם קיימות.

**3. הוספת תגית לקובץ**

* **Route**:  
  POST /api/files/{fileId}/tags
* **פרמטרים**:  
  **Body**: { tag: string }  
  **Headers**: Authorization: Bearer <JWT\_TOKEN>
* **פלט**:  
  data: { fileId: string, tags: string[] }  
  **קודי סטטוס**: 200 OK, 400 Bad Request (במקרה של תגית לא חוקית)
* **לוגיקה עסקית**:
  + הוספת תגית למערכת.
  + עדכון רשימת התגיות עבור הקובץ.

**4. חיפוש קבצים לפי תגית**

* **Route**:  
  GET /api/files/search
* **פרמטרים**:  
  **Query Params**: tags=tag1,tag2  
  **Headers**: Authorization: Bearer <JWT\_TOKEN>
* **פלט**:  
  data: { files: File[] }  
  **קודי סטטוס**: 200 OK, 404 Not Found (אם לא נמצאו קבצים עם התגיות המבוקשות)
* **לוגיקה עסקית**:  
  סינון הקבצים על פי תגיות שנבחרו, הצגת תוצאות ממוקדות.

**3.2 אימות והרשאות**

המערכת מבצעת אימות באמצעות **JWT Authentication**.

* **תהליך האימות**:
  + המשתמשים מבצעים התחברות עם דוא"ל וסיסמה.
  + לאחר התחברות מוצלח, המערכת מחזירה **JWT Token** אשר יש לשלוח בהמשך לכל בקשה שדורשת הרשאות.
* **תפקידים וההרשאות**:
  + **משתמש רגיל**:
    - יכול להעלות קבצים.
    - יכול להוסיף תגיות לקבצים.
    - יכול לראות ולהוריד את הקבצים האישיים שלו.
  + **מנהל מערכת (Admin)**:
    - יכול לנהל את כל המשתמשים במערכת (הוספה, עדכון, מחיקה).
    - יכול להפיק דוחות וסטטיסטיקות פעילות.
    - יכול להגדיר הגדרות מערכת כמו מכסות אחסון.

**3.3 אפליקצית ניהול**

**יכולות המנהל:**

* **דוחות**:
  + הפקת **דוחות פעילות משתמשים**: המנהל יכול להפיק דוחות שמציגים את הפעילויות האחרונות של כל משתמש, כמו העלאת קבצים והוספת תגיות.
  + הפקת **סטטיסטיקות שימוש**: מידע על מספר המשתמשים הפעילים, מספר הקבצים שהועלו, ומגבלות השימוש.
* **ניהול משתמשים**:
  + המנהל יכול לבצע **CRUD (יצירה, קריאה, עדכון, מחיקה)** עבור המשתמשים.
  + יכול לשייך **תפקידים** למשתמשים (כגון: משתמש רגיל או מנהל).
* **הרשאות**:
  + המנהל יכול לעדכן **הרשאות למשתמשים**, למשל להוסיף או להסיר אפשרות להעלות קבצים או לגשת לדוחות.
* **הגדרות מערכת**:
  + המנהל יכול לנהל **פרמטרים גלובליים** כמו מכסות אחסון לכל משתמש.
  + יכול להגדיר **מגבלות גודל קובץ** והגדרות אבטחה.

**4. תרשים מבנה הטבלאות (Database Schema)**

**1. טבלת Users**

טבלת המשתמשים מכילה את כל המידע על המשתמשים במערכת.

|  |  |  |  |
| --- | --- | --- | --- |
| **שם העמודה** | **סוג נתונים** | **מאפיינים** | **תיאור** |
| id | INT | PK | מזהה ייחודי |
| first\_name | VARCHAR(50) | NOT NULL | שם פרטי |
| last\_name | VARCHAR(50) | NOT NULL | שם משפחה |
| email | VARCHAR(100) | UNIQUE, NOT NULL | אימייל |
| password | VARCHAR(255) | NOT NULL | סיסמא (מאובטח) |
| role | VARCHAR(20) | NOT NULL | תפקיד (משתמש רגיל/מנהל) |
| created\_at | DATETIME | DEFAULT NOW() | תאריך יצירה |
| created\_by | INT | FK (Users) | נוצר על ידי (ID של המשתמש) |
| updated\_at | DATETIME | DEFAULT NOW() | תאריך עדכון |
| updated\_by | INT | FK (Users) | עודכן על ידי (ID של המשתמש) |

**2. טבלת Files**

טבלת הקבצים מכילה את המידע על הקבצים שהעלו המשתמשים.

|  |  |  |  |
| --- | --- | --- | --- |
| **שם העמודה** | **סוג נתונים** | **מאפיינים** | **תיאור** |
| id | INT | PK | מזהה ייחודי |
| user\_id | INT | FK (Users) | מזהה המשתמש שהעלה את הקובץ |
| file\_name | VARCHAR(255) | NOT NULL | שם הקובץ |
| file\_path | VARCHAR(500) | NOT NULL | נתיב הקובץ בשרת |
| file\_size | INT | NOT NULL | גודל הקובץ (בבייטים) |
| file\_type | VARCHAR(50) | NOT NULL | סוג הקובץ (למשל: pdf, jpg) |
| tags | TEXT | NULL | תגיות עבור הקובץ (אופציונלי) |
| created\_at | DATETIME | DEFAULT NOW() | תאריך יצירה |
| created\_by | INT | FK (Users) | נוצר על ידי (ID של המשתמש) |
| updated\_at | DATETIME | DEFAULT NOW() | תאריך עדכון |
| updated\_by | INT | FK (Users) | עודכן על ידי (ID של המשתמש) |

**3. טבלת Tags**

טבלת התגיות מכילה את כל התגיות שניתן להוסיף לקבצים.

|  |  |  |  |
| --- | --- | --- | --- |
| **שם העמודה** | **סוג נתונים** | **מאפיינים** | **תיאור** |
| id | INT | PK | מזהה ייחודי |
| tag\_name | VARCHAR(50) | UNIQUE, NOT NULL | שם התגית |
| created\_at | DATETIME | DEFAULT NOW() | תאריך יצירה |
| created\_by | INT | FK (Users) | נוצר על ידי (ID של המשתמש) |
| updated\_at | DATETIME | DEFAULT NOW() | תאריך עדכון |
| updated\_by | INT | FK (Users) | עודכן על ידי (ID של המשתמש) |

**4. טבלת File\_Tags (לניהול קשרים Many-to-Many בין קבצים ותגיות)**

|  |  |  |  |
| --- | --- | --- | --- |
| **שם העמודה** | **סוג נתונים** | **מאפיינים** | **תיאור** |
| file\_id | INT | FK (Files) | מזהה הקובץ |
| tag\_id | INT | FK (Tags) | מזהה התגית |
| created\_at | DATETIME | DEFAULT NOW() | תאריך יצירה |
| created\_by | INT | FK (Users) | נוצר על ידי (ID של המשתמש) |

הטבלה הזו מנהלת את הקשרים בין קבצים ותגיות, כלומר כל קובץ יכול להיות משויך למספר תגיות, וכך גם התגיות יכולות להיות משויכות למספר קבצים.

**הקשרים בין הטבלאות**

* **One-to-Many**:
  + **Users** → **Files**: כל משתמש יכול להעלות מספר קבצים, אך כל קובץ שייך למשתמש אחד.
  + **Users** → **Tags** (לא ישירות, אלא דרך **File\_Tags**): כל תגית נוצרה על ידי משתמש, וניתן לקשר אותה לקבצים דרך טבלת החיבור.
* **Many-to-Many**:
  + **Files** ↔ **Tags**: כל קובץ יכול להיות משויך למספר תגיות, וכל תגית יכולה להיות משויכת למספר קבצים. הקשר הזה מנוהל דרך טבלת **File\_Tags**.

**ספרינט 1 (שבוע 1)**

**מטרה**: הקמת תשתית בסיסית + פיתוח תהליך התחברות והרשמה

* יצירת פרויקטים ב-.NET, React, ו-Angular.
* בניית מסד נתונים בסיסי עם טבלאות משתמשים.
* פיתוח API להתחברות והרשמה עם JWT.
* פיתוח מסך התחברות ב-React ו-Angular.
* התחברות למסד נתונים.

**ספרינט 2 (שבוע 2)**

**מטרה**: פיתוח ניהול קבצים ואלבומים

* פיתוח API להעלאת קבצים (בצד שרת).
* יצירת מסך ניהול קבצים ב-React/Angular.
* פיתוח יכולת שיתוף קבצים וקטגוריות.
* אינטגרציה של מסך ניהול קבצים עם ה-API.

**ספרינט 3 (שבוע 3)**

**מטרה**: פיתוח מערכת ניהול משתמשים + פיצ'רים נוספים

* פיתוח API לניהול משתמשים (CRUD).
* יצירת מסך ניהול משתמשים ב-React/Angular.
* פיתוח מערכת הרשאות למשתמשים.
* שיפור חווית המשתמש (UI/UX) למסכים הקיימים.

**ספרינט 4 (שבוע 4)**

**מטרה**: בדיקות, תיקונים ופריסה

* ביצוע בדיקות פונקציונליות על המערכת כולה.
* תיקוני באגים ושיפור ביצועים.
* פריסת המערכת בסביבת שרת AWS.
* הגדרת דומיין אישי.
* עיצוב דף הבית לשיווק.