
# תיק משימה – גלריה גרסה V1.0.2

## רקע ותכולת עבודה לגרסה

הגרסאות הראשונות של "גלריה" היו מוצלחות ועבדו בצורה טובה, אולם אין באפשרותה לשמור מידע בין הריצות השונות ☹.

מבחינה טכנולוגית, בפרויקט "גלריה" אנו עושים שימוש בנתונים ובמידע מסוגים שונים – אלבומים, תמונות, משתמשים ותיוגים. כרגע, המידע לא נשמר ומאוחסן ב-**RAM** של המחשב, כלומר, בכל פעם שהאפליקציה נסגרת המידע נמחק. זוהי כמובן, התנהגות לא רצויה.

הדרישה היא שמגרסה V1.0.2 הנתונים ישמרו בצורה קבועה (או פֶּרְסִיסטֶנסית - מהמילה האנגלית **Persistence**), כלומר נרצה לשמור את הנתונים ללא תלות בריצת האפליקציה בצורה קבועה ונוחה. הדבר יתבצע על ידי שימוש ב-**Database** (מסד הנתונים).

## שלבי עבודה

### שלב 0: הוספת משימות (Issue-ים)

כמו שאמרנו, המשימה שלנו היא - שמירת נתוני אלבומים ותמונות בין הרצה להרצה של האפליקציה.

א. **משימת תשתית:** פצלו את המשימה הזו לארבע משימות קטנות (בהתאם לארבעת השלבים המפורטים בהמשך) ועם תאריכי יעד לפי הנחיות המדריך, וצרו להם **Issue**-ים חדשים ב-**GitLab**.

**וודאו לעדכן את הסטטוס של המשימות אחרי כל סיום שלב.**

### שלב 1: יצירת תשתית חיבור למסד הנתונים

א. **משימת לימוד:** הכרות עם מסדי נתונים ושפת SQL -
**Lab1\Learning Task 11 - DB&SQL**

**ב.** **משימת לימוד:** המודל הטבלאי –
**Lab1\Learning Task 12 - Table-Model**

**ג.** **משימת לימוד:** חיבור ל-SQLite ב-++C -
**Lab1\Learning Task 13-C++-SQLite**

ד. **משימת פיתוח:** יצירת מחלקת **DatabaseAccess** וביצוע משימות **שלב 1** במימוש של מתודות מחלקת IDataAccess.

ה. **משימת תשתית:** סמנו את ה-issue כ-Done.

## שלב 2: מימוש פעולות עדכון נתונים

א. **משימת לימוד:** פעולות עדכון נתונים – **Lab2\Learning Task 21-SQL**

ב. **משימת לימוד:** חיבור בין SQLite ל-++C לעדכון נתונים – **Lab2\Learning Task 22-C++-SQLite**

ג. **משימת פיתוח:** ביצוע משימות **שלב 2** במימוש של מתודות מחלקת **IDataAccess**.

ד. **משימת תשתית:** סמנו את ה-issue כ-Done.

## שלב 3: מימוש פעולות שליפת נתונים

א. **משימת לימוד:** חיבור בין SQLite ל-++C לעדכון נתונים – **Lab3\Learning 31-C++-SQLite**

ב. **משימת פיתוח:** ביצוע משימות **שלב 3** במימוש של מתודות מחלקת **IDataAccess**.

ג. **משימת תשתית:** סמנו את ה-issue כ-Done.

## שלב 4: רשות – בדיקות למחלקת הבסיס

א. **משימת בדיקות:** בדיקות למחלקת הבסיס **IDataAccess**. פירוט נמצא ב-נספח: בדיקות למחלקת הבסיס IDataAccess.

ב. **משימת תשתית:** אם ביצעתם את המשימה הזו - סמנו את ה-issue כ-Done. אחרת – מחקו אותו.

## שלב 5: העשרה (מתוך Lab4)

א. **משימת העשרה:** מודל השכבות למערכת תוכנה.

ב. **משימת העשרה:** נרמול נתונים.

המשימות האלו נמצאות בתיקייה **Lab4**.

# Design לגרסה

הוסיפו מחלקה בשם **DatabaseAccess** אשר תירש מהמחלקה **IDataAccess** ותממש את כל המתודות הווירטואליות המוגדרות בה (תיאור המחלקה יפורט בהמשך המסמך).

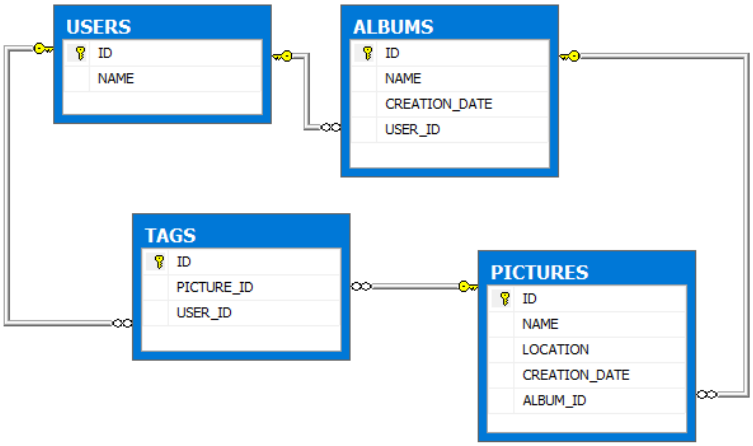
המחלקה **DatabaseAccess** תעשה את כל מה שעושה המחלקה **MemoryAccess**, אולם המידע שהיא תנהל ישמר במסד נתונים מסוג **SQLite**.

**שימו לב!** תצטרכו לשנות בקובץ **Gallery.cpp** את הטיפוס של המשתנה **dataAccess**.

אם תרצו להעמיק במבנה של פרויקט גלריה, תוכל לקרוא את המאמר המצורף - על **Application Structure** ב- **Lab4**.

## מחלקות ה-Data Access Layer

מחלקות אלו מייצגות את מבנה הנתונים כפי שהוא מוגדר במסד הנתונים.
הטבלאות מיוצגות באמצעות מחלקות (Entities):



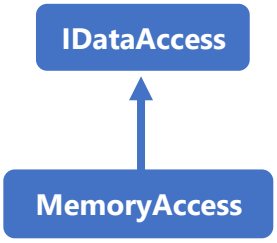

- המחלקה **User** - מייצגת רשומה מהטבלה **USERS**.
- המחלקה **Album** – מייצגת רשומה מהטבלה **ALBUMS**.
- המחלקה **Picture** – מייצגת רשומה מהטבלה **PICTURES**.

## המחלקה MemoryAccess

מחלקה זו מטפלת בניהול המידע הנמצא בזיכרון המחשב בזמן ריצת האפליקציה.

**זו המחלקה הממומשת כרגע ואנו נחליף אותה.**

מחלקה זו יורשת מהממשק **IDataAccess** אשר מגדיר את כל הפעולות האפשריות על המידע של האפליקציה באמצעות מתודות וירטואליות.



## המתודות של מחלקת הבסיס IDataAccess

**חשוב!** שימו לב למספרים המופיעים לצד חלק מהתיאורים של המתודות. הם מפנים להערות חשובות המופיעות בהמשך.

| חתימת הפונקציה | תיאור הפונקציה | שלב |
|---|---|---|
| `open();` | פתיחת מסד נתונים | 1 |
| `close();` | סגירת מסד נתונים | 1 |
| `clear();` | מנקה את כל המידע ששמור במחלקה [2] | 1 |
| **Album** | | |
| `getAlbums();` | שליפת כל האלבומים [1] | 3 |
| `getAlbumsOfUser ( const User& user );` | החזרת כל האלבומים של משתמש מסוים [1] | 3 |
| `createAlbum ( Album& album );` | יצירת אלבום חדש | 3 |
| `deleteAlbum ( const std::string& albumName, int userId );` | מחיקת אלבום קיים | 2 |
| `doesAlbumExists ( const std::string& albumName, int userId );` | בדיקה האם קיים אלבום בעל השם | 3 |
| `openAlbum ( const std::string& albumName );` | פתיחת אלבום [1] | 3 |
| `closeAlbum ( Album& album );` | סגירת אלבום | 1 |
| `printAlbums();` | הדפסת כל האלבומים | 3 |
| **Picture** | | |
| `addPictureToAlbumByName ( const std::string& albumName, const Picture& picture );` | הוספת תמונה לאלבום | 3 |
| `removePictureFromAlbumByName ( const std::string& albumName, const std::string& pictureName );` | הסרת תמונה מאלבום | 3 |

| | | |
|---|---|---|
| `tagUserInPicture ( const std::string& albumName, const std::string& pictureName, int userId );` | הוספת תיוג לתמונה | 2 |
| `untagUserInPicture ( const std::string& albumName, const std::string& pictureName, int userId );` | הסרת תיוג מתמונה | 2 |
| **User** | | |
| `printUsers();` | הדפסת כל המשתמשים שקיימים | 3 |
| `getUser ( int userId );` | מחזירה משתמש על פי ID [1] | 3 |
| `createUser ( User& user );` | יצירת משתמש למערכת | 2 |
| `deleteUser ( const User& user );` | מחיקת משתמש | 2 |
| `doesUserExists ( int userId );` | בדיקה האם משתמש קיים לפי **userid** | 3 |
| **Statistics** | | |
| `countAlbumsOwnedOfUser ( const User& user );` | סופרת כמה אלבומים יש למשתמש | 3 |
| `countAlbumsTaggedOfUser ( const User& user );` | סופרת כמה תיוגים יש למשתמש בכל תמונות האלבום | 3 |
| `countTagsOfUser ( const User& user );` | סופרת כמה תיוגים יש למשתמש בתמונה | 3 |
| `averageTagsPerAlbumOfUser ( const User& user );` | ממוצע תיוגים למשתמש באלבום | 3 |
| **Queries** | | |
| `getTopTaggedUser();` | קבלת המשתמשים המתויגים ביותר | 3 |
| `getTopTaggedPicture();` | קבלת התמונות המתויגות ביותר | 3 |
| `getTaggedPicturesOfUser ( const User& user );` | קבלת התמונות המתויגות במשתמש מסוים [1] | 3 |

↑ חזרו למעלה ↑

**שימו לב:** ההערות נמצאות בדף הבא.

**הערות:**

**[1]** - אלו מתודות שמחזירות פוינטר/רפרנס לאובייקט מסויים (תמונה, משתמש, אלבום או מערך שלהם). האחריות על האובייקט שאליו מפנה הפוינטר/רפרנס המוחזר הוא על המחלקה שאחראית על הנתונים, במקרה שלנו **DatabaseAccess**, המחלקה שאותם אתם כותבים.

כלומר, אתם צריכים במחלקה **DatabaseAccess** ליצור את האוביקטים האלה, ולתת את הפוינטר / רפרנס אליהם.

**[2]** - הכוונה למחיקה ושחרור כל הזיכרון שהוקצה ע"י האובייקטים המתוארים ב-**[1]**.

## תרגול מעשי – הוספת DB לגלריית התמונות

**אופן ההגשה:**

- המשיכו לעבוד על ה- repo של פרויקט גלריה שיצרתם ושיתפתם עם המדריך בשיעור "ניהול קוד".
- המשיכו לעבוד על אותו פרויקט Visual Studio משלבי הלמידה.
- צרו branch בשם **feature/DB_Abillities** היוצא מ-**develop** .
- **שימו לב!** בצעו commit-ים לאורך כל התהליך! דאגו לרשום להם הודעות אינדיקטיביות ומובנות.
- כאשר סיימתם - וודאו שהפרויקט מוכן להגשה (מתקמפל ועובד טוב) בתוך ה-branch שפתחתם - **feature/DB_Abillities**.
- כנסו ל-repo באתר GitLab ובצעו Merge Request מ- **feature/DB_Abillities** אל תוך **develop.**

עליכם למשש את המתודות המופיעות בטבלה מעל לפי כל שלב בתכנית העבודה.

**טיפ: השאירו את הפונקציות שכוללות שליפה לסוף המימוש. בדקו שהקוד שלכם עובד כמו שצריך בעזרת sqlite ב-cmd כמו שעשיתם בשיעור הקודם.**

## נספח: בדיקות למחלקת הבסיס IDataAccess

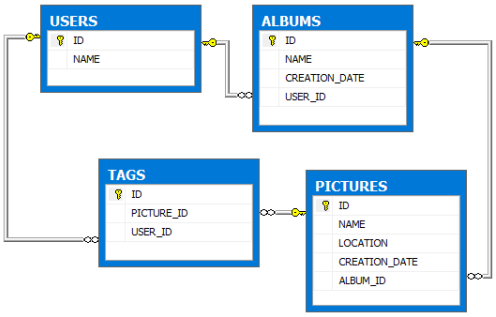
אחד השלבים בפיתוח תוכנה הוא בדיקה שלה ווידוא שהדברים עובדים כמתוכנן.

בדיקות תוכנה הוא נושא מורכב מאוד וישנם סוגי בדיקות רבים. אנחנו נוסיף לפרויקט שלנו מספר בדיקות בסיסיות, המאפשרות לזהות במהירות אם התוכנה עובדת כנדרש. בדיקות כאלו נקראות בדיקות שפיות (Sanity Test).

כמו כל המפתחים הטובים בתעשייה לא יצא מוצר לשוק לפני שביצענו לו מספר בדיקות (כרגע זה בסיסי, בעתיד תלמדו יותר לעומק את הנושא ☺ )

- המשיכו לעבוד על ה- repo של פרויקט גלריה שיצרתם ושיתפתם עם המדריך בשיעור "ניהול קוד".
- המשיכו לעבוד על אותו פרויקט Visual Studio משלבי הלמידה.
- צרו branch בשם **feature/DB_Testing** היוצא מ **develop** .

צרו מחלקה חדשה בתוך הפרויקט בשם **DataAccessTest.cpp** (אל תשכחו גם h. ) אשר תכיל את היכולות הבאות:




## בניית טבלאות -

בנו באמצעות קוד בשפת ++C את מסד הנתונים של Gallery:

- לאחר שסיימתם בצעו commit אינפורמטיבי לתוך **feature/DB_Testing**.

## הוספת רשומות -

הזינו ב- Gallery שלושה משתמשים לטבלה **USERS**.
לכל אחד מהמשתמשים יהיה אלבום אחד המכיל שתי תמונות,
בכל אחת מהתמונות יופיעו לפחות 2 תיוגים של משתמש רשום.

- לאחר שסיימתם בצעו commit אינפורמטיבי לתוך **feature/DB_Testing**.

## עדכון רשומות –

אחד המשתמשים העלה תמונה וכתב ששמה הוא "My Femily", כמה שניות לאחר מכן הוא גילה את שגיאת הכתיב.
העלו תמונה עם שגיאת הכתיב ולאחר מכן תקנו את הטעות.

- לאחר שסיימתם בצעו commit אינפורמטיבי לתוך **feature/DB_Testing**.

## מחיקת רשומות -

מחקו את אחד המשתמשים, את הגלריות והתמונות שלו.

- לאחר שסיימתם בצעו commit אינפורמטיבי לתוך **feature/DB_Testing**.
- וודאו שהפרויקט מוכן להגשה (מתקמפל ועובד טוב) בענף **feature/DB_Testing**.
- כנסו ל-repo באתר GitLab ובצעו Merge Request מ- **feature/DB_Testing** אל תוך **develop**.

↑ חזרו למעלה ↑