**Documentation**

האפליקציה מתחלקת ל-3 שכבות: PL,BL,DAL

נתחיל עם שכבת ה-PL – שכבת הצגת הנתונים ובמקרה שלנו ה-SWAGGER+הקונטרולר

[HttpPost]

public IActionResult PostFamilyTree(List<FamilyMemberParent> FamilyMemberParents)

{

Log.Logger = new LoggerConfiguration()

.WriteTo.File("log.txt", rollingInterval: RollingInterval.Infinite)

.CreateLogger();

בקונטרולר שלנו נעשה שימוש בפונקציית POST אחת לשליחת ה-INPUT ויצרנו אובייקט לוג חדש לכתיבת הלוגים של האפליקציה.

try

{

Log.Information("Application started");

List<FamilyMemberParent> peopleList = FamilyMemberParents.OrderBy(p => p.Parent).ToList();

int nullsCount = peopleList.Count(item => item.Parent == null);

var duplicates = peopleList.GroupBy(x => x.Id)

.Where(g => g.Count() > 1)

.Select(g => g.Key);

var idSet = new HashSet<int>(peopleList.Select(obj => obj.Id));

bool allParentsExist = true;

foreach (var obj in peopleList)

{

if (obj.Parent.HasValue && !idSet.Contains(obj.Parent.Value))

{

allParentsExist = false;

break;

}

}

בשלב זה אנחנו מנסים לתפוס את כל מקרי הקיצון האפשריים שעלולים לגרום לאפליקציה שלנו לקרוס.

בדקנו האם ה-INPUT מכיל :

1. אובייקטים עם אותה תעודת זהות
2. אובייקטים עם ערכים ריקים
3. יותר מאובייקט מקור אחד (אובייקט מקור מוגדר כחסר אבא –NULL)
4. פחות מאובייקט מקור אחד (אובייקט מקור מוגדר כחסר אבא –NULL)
5. אובייקט המכיל מספר אבא שלו קיים ברשימה

if (nullsCount > 1)

throw new ArgumentNullException($"There are {nullsCount} nulls in the list, only 1 object can contain null");

if (nullsCount < 1)

throw new ArgumentNullException($"At least 1 object has to contain null in his parentId property");

if (duplicates.Any())

throw new DuplicateWaitObjectException($"There are {duplicates} objects with the same Id");

if (!allParentsExist)

throw new FormatException("Parent Id not found");

במתודולוגיית TRY ו-CATCH אנו זורקים את השגיאות המתאימות עבור כל מקרה ומנטרים אותן להודעות למשתמש+כתיבת סטטוס הפעולה בקובץ הלוג.

catch (ArgumentNullException ex)

{

Log.Error(ex.ParamName);

return BadRequest(ex.ParamName);

}

catch (DuplicateWaitObjectException ex)

{

Log.Error(ex.ParamName);

return Conflict(ex.ParamName);

}

catch (FormatException ex)

{

Log.Error(ex.Message);

return BadRequest(ex.Message);

}

catch (Exception ex)

{

Log.Error(ex.Message);

return BadRequest(ex.Message);

}

לבסוף במידה ואין שגיאות נעבור הלאה לשכבת ה-BL ובמידה והתהליך הושלם בהצלחה, נחזיר סטטוס 200 עם התשובה.

FamilyMemberParent familyMember = new FamilyMemberParent();

var formatedTree = familyMember.familyMembersFunc(peopleList);

Log.Information("Application finished");

return Ok(formatedTree);

השכבה השנייה היא שכבת ה-BL המכילה את המחלקות ואת הגדרותיהם בנוסף להגדרת ולידציות ואילוצים על השדות.

באפליקציה שלנו ישנם 2 מחלקות, מחלקת ילדים המכילה את השדות: מספר תעודת זהות, שם מסוג STRING ורשימת ילדים מסוג "ילדים". יצרנו בנאים למחלקה וערכים דיפולטיביים.

public class FamilyMemberChildren

{

public int Id { get; set; }

public string Name { get; set; }

public List<FamilyMemberChildren> Childs { get; set; }

public FamilyMemberChildren(int id, string name, List<FamilyMemberChildren> childs)

{

Id = id;

Name = name;

Childs = childs;

}

public FamilyMemberChildren() {

Id = 0;

Name = "Default Name";

Childs = new List<FamilyMemberChildren>();

}

}

המחלקה השנייה היא מחלקת הורים המכילה את השדות: מספר תעודת זהות, שם מסוג STRING ומספר תעודת זהות של הורה + ולידציות ואילוצים: : מספר תעודת זהות הוא חובה, שם האובייקט הוא חובה והוא חייב לכלול לפחות אות אחת בשם, מספר תעודת זהות של הורה חייב להיות מספר או NULL.

[Required(ErrorMessage ="Every object must have an id")]

public int Id { get; set; }

[Required(ErrorMessage = "Every object must have a name")]

[MinLength(1,ErrorMessage = "Every object name must contain at least 1 letter")]

public string Name { get; set; }

public int? Parent { get; set; }

הגדרנו בנאים ופונקציית מעבר לשכבת ה-DAL:

public FamilyMemberParent(int id, string name, int parent)

{

Id = id;

Name = name;

Parent = parent;

}

public FamilyMemberParent() { }

public List<FamilyMemberChildren> familyMembersFunc(List<FamilyMemberParent> peopleList)

{

FamilyTreeDAL familyTree = new FamilyTreeDAL();

return familyTree.createTree(peopleList);{

השכבה האחרונה היא שכבת ה-DAL- שם מבצעים את כל האינטרקציות עם ה-DB ובמקרה שלנו פותרים את השאלה:

הרעיון הוא להשתמש באלגוריתם BFS כך שבכל שלב, אנחנו מוצאים את הענפים הקרובים ביותר לענף המקור ובודקים האם קיים יחס של אבא וילד בין הענף הקרוב לענף האבא.

בשלב הראשון אנחנו נגדיר משתנים שיעזרו לנו בפתרון השאלה:

List<FamilyMemberChildren> list = new List<FamilyMemberChildren>();

FamilyMemberChildren mainTree = new FamilyMemberChildren();

mainTree.Id = peopleList[0].Id;

mainTree.Name = peopleList[0].Name;

list.Add(mainTree);

int i = 1;

הגדרנו רשימה של אובייקטים מסוג "ילדים", ענף מקור שמאותחל באיבר הראשון מה-INPUT, הוספנו את ענף המקור לרשימה שתשמש אותנו בהמשך למציאת "הילדים" של ענף המקור. (בנוסף אתחלנו את האינדקס הרץ ל-1, כאמור השתמשנו באיבר הראשון מה-INPUT כדי לאתחל את השדות של ענף המקור).

בשלב השני אנחנו נכנסים ללולאה:

while(i < peopleList.Count)

{

FamilyMemberChildren Node = list.First();

if (peopleList[i].Parent==Node.Id)

{

FamilyMemberChildren newChild = new FamilyMemberChildren();

newChild.Id = peopleList[i].Id;

newChild.Name = peopleList[i].Name;

Node.Childs.Add(newChild);

list.Add(newChild);

}

else

{

i--;

list.RemoveAt(0);

}

i++;

}

בכל אינטרציה אנחנו בודקים אם כבר סיימנו לשרשר את כל האנשים כילדים לאבות שלהם.

במידה וכן, אנחנו עובדים ולשלב השלישי.

במידה ולא, אנחנו נכנסים ללולאה ויוצרים משתנה חדש שמאותחל כענף המקור ואז מגיעים לשאלה:

האם הענף שלנו הוא האבא של האיבר מה-INPUT?

במידה וכן, אנחנו ניצור אובייקט "ילד" חדש, נאתחל אותו בשדות של האיבר מה-INPUT ונכניס אותו כ-"ילד חדש" לרשימת הילדים של הענף שלנו.

לבסוף נכניס את הילד החדש גם לרשימה למציאת הילדים של "ענף המקור" ונקדם את האינדקס באחד.

\*נשאלת השאלה - מדוע אנחנו מכניסים גם את הילד החדש לרשימת הענפים? או לכתחילה מדוע אנחנו יוצרים אובייקט "ילד" חדש ומעדכנים את הענף המשני שלנו ולא ישירות את ענף המקור?

הסיבה לכך היא שאנחנו לא בטוחים תמיד שהאובייקט הבא ברשימה הוא "ילד ישיר" של ענף המקור, וכדי למנוע דריסת נתונים על ענף המקור, אנחנו יוצרים בכל שלב ענף משני שבמידה והוא לא ילד ישיר של האבא, אנחנו נוכל לנטר את הענפים ולהגיע לאבא המתאים של אובייקט הילד.

במידה ו-הענף שלנו הוא האבא של האיבר מה-INPUT, אנחנו לא נתקדם ברשימת האיברים מה-INPUT כדי למצוא את האבא המתאים של הילד, ונוציא מהרשימה את האיבר הראשון.

אנחנו יכולים להוציא את האיבר הראשון מבלי לחשוש שיהיו לו עוד ילדים מבין האיברים שמגיעים מה-INPUT וזאת משום שמיינו את רשימת האיברים בסדר עולה על פי מספר הזהות של האבא שלהם, ואם מספר הזהות של האבא אצל הילד הוא לא כמספר הזהות של האבא שלנו אזי בוודאות כל הילדים של האבא שלנו נגמרו ואפשר להתקדם לאבא הבא ולמצוא את הילדים שלו.

בשלב השלישי אנחנו נהפוך את ענף המקור שלנו לאובייקט מסוג רשימה ונחזיר אותו כ-OUTPUT.

List<FamilyMemberChildren> mainTreeList = new List<FamilyMemberChildren>() { mainTree };

return mainTreeList;