**Coding Test.**

**Model Structures**

// Basic classes

public abstract class Node

{

public string Name { get; set; } = "";

public Folder Parent { get; set; }

// Base object identifiers

public int folder\_id { get; set; } = -1;

public int item\_id { get; set; } = -1;

}

// Folder object

public class Folder : Node

{

public List<Node> Children { get; set; }

public Folder()

{

Children = new List<Node>();

}

}

// Item object

public class Item : Node

{

}

**Methods and functions:**

// Main object Tree

public class Tree

{

public Folder Root { get; private set; }

// Sequence/counter to for base object identifiers

public int folder\_index { get; set; }

public int item\_index { get; set; }

// Initialising Root methods

public Tree(string rootName)

{

Root = new Folder { Name = rootName };

Root.folder\_id = 0;

folder\_index = 0;

}

// Add Item function

public Item AddItem(Folder parent, string itemName)

{

var item = new Item { Name = itemName, Parent = parent };

item\_index += 1;

item.item\_id += item\_index;

parent.Children.Add(item);

return item;

}

// Delete Item method

public void DeleteItem(Item item)

{

item.Parent.Children.Remove(item);

}

// Add Folder function

public Folder AddFolder(Folder parent, string folderName)

{

var folder = new Folder { Name = folderName, Parent = parent };

folder\_index += 1;

folder.folder\_id = folder\_index;

parent.Children.Add(folder);

return folder;

}

// Delete Folder method including Childen objects

public void DeleteFolder(Folder folder)

{

// Hanling for root

if (folder.folder\_id == 0)

{

folder.Children.Clear();

return;

}

folder.Parent.Children.Remove(folder);

}

// Moving objects method

public void Move(Node node, Folder newParent)

{

node.Parent.Children.Remove(node);

node.Parent = newParent;

newParent.Children.Add(node);

}

// Serchig function not used for this project

public List<Node> Search(string name, Folder startFolder)

{

List<Node> results = new List<Node>();

foreach (var child in startFolder.Children)

{

if (child.Name.Contains(name))

results.Add(child);

if (child is Folder childFolder)

results.AddRange(Search(name, childFolder));

}

return results;

}

// Additional sercing function - Get Item By Index

public Item GetItemByIndex(int index, Folder startFolder)

{

Item ret = new Item();

foreach (var child in startFolder.Children)

{

if (child is Item && child.item\_id == index)

return (Item)child;

if (child is Folder childFolder)

{

ret = GetItemByIndex(index, childFolder);

if (ret is Item && ret.item\_id == index)

return (Item)ret;

}

}

return ret;

}

// Additional sercing function - Get Folder By Index

public Folder GetFolderByIndex(int index, Folder startFolder)

{

Folder ret = new Folder();

if (index == 0)

return startFolder;

foreach (var child in startFolder.Children)

{

if (child is Folder && child.folder\_id == index)

return (Folder)child;

if (child is Folder childFolder)

{

ret = GetFolderByIndex(index, childFolder);

if (ret is Folder && ret.folder\_id == index)

return (Folder)ret;

}

}

return ret;

}

//Function is returning curren Tree in List<string>

//Structure of string:

//1 - Leading spaces depend on hierarchy level

//Symvol F/I = Item/Foolder + Symvol\_space

//numeric ID = Item/Foolder identifier + Symvol\_space

//Text = Object Name

public List<string> PrintList(Folder startFolder,int level)

{

List<string> results = new List<string>();

string str;

if (level == 0)

results.Add("F 0 ROOT");

level ++;

foreach (var child in startFolder.Children)

{

if (child is Folder childFolder)

{

str = new String(' ', level \* 4) + "F " + child.folder\_id + " " + child.Name;

results.Add(str);

results.AddRange(PrintList(childFolder, level + 1));

}

else

{

str = new String(' ', level \* 4) + "I " + child.item\_id + " " + child.Name;

results.Add(str);

}

}

return results;

}

}

**Example Usage:**

Public static void Main()

{

Tree tree = new Tree("Root");

// Add folders into ROOT

tree.AddFolder(tree.Root, "Folder1");//Foder\_ID = 1

tree.AddFolder(tree.Root, "Folder2");//Foder\_ID = 2

tree.AddFolder(tree.Root, "Folder3");//Foder\_ID = 3

// Add subfolders into Folder where Folder.Foder\_ID = 1

var f4 = tree.AddFolder((Folder) tree.GetFolderByIndex(1, tree.Root), "subfolder");//Foder\_ID = 4

// Add item into subfolder

tree.AddItem(f4, "Item2");

// or tree.AddItem(tree.GetFolderByIndex(4, tree.Root), "Item2");

// Add items into root

tree.AddItem(tree.Root, "Item1");

// Add item into subfolder

tree.AddItem(tree.GetFolderByIndex(4, tree.Root), "Item2\_1");

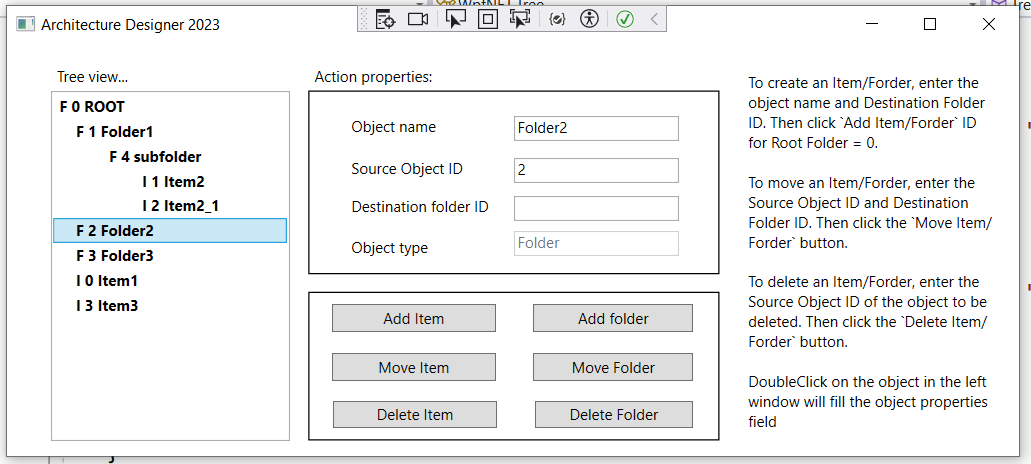
// Search

//var searchResults = tree.Search("Item", tree.Root);

refresh\_list();

}

**The project was done as a WPF Solution.**



Form event handling, WPF.xaml and validity checks can be viewed in the project archive as a whole or from [here](https://drive.google.com/file/d/1prx8R7R_gpVbqsHLXw3bzddQcnyp2TYo/view?usp=sharing). The form also contains tooltips in the objects for user actions. You can watch a video on how the app works [here](https://youtu.be/yl-GdpozpBY).

**Realization of recursion by means of database (Oracle)**

**ErModel, structure, SQL Script:**

|  |  |  |
| --- | --- | --- |
|  |  | CREATE TABLE Folder(  id NUMBER(19, 0) Generated Always AS IDENTITY (START WITH 1 INCREMENT BY 1 CACHE 20 NOCYCLE),  Parent\_ID NUMBER(19, 0),  Name CHAR(255) NOT NULL,  TRUE\_Folder NUMBER(1, 0) NOT NULL,  Description VARCHAR2(255),  Item\_Link VARCHAR2(512),  CONSTRAINT PK1 PRIMARY KEY (id)  );  ALTER TABLE Folder ADD CONSTRAINT RefFolder71  FOREIGN KEY (Parent\_ID)  REFERENCES Folder(id); |

**Table Folder with folders and items:**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID (Identity/Sequence)** | **Parent\_ID (Null = Root)** | **Name** | **TRUE\_FOLDER** |
| 1 | Null | Folder1 | True |
| 2 | Null | Folder2 | True |
| 3 | Null | Item1 | False |
| 4 | 1 | Inner\_Folder1 | True |
| 5 | 4 | Inner\_Item1 | False |
| 6 | 4 | Inner\_Folder2 | True |
| 7 | 6 | Inner\_Item3 | False |

**Distribution of objects in the information structure:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Level 0 (ROOT)** | **Level 2** | **Level 3** | **Level 4** |
| Folder1 | Inner\_Folder1 | Inner\_Folder2 | Inner\_Item3 |
| Inner\_Item2 | - |
| Inner\_Item1 | - |  |
| Folder2 |  |  |  |
| Item1 | - |  |  |

Data integrity is ensured by database means. Row deletion is blocked if there is a descendant in the hierarchy.

Moving brunch “Inner\_Folder1” from “Folder1” into “Folder2”, use next script:

|  |
| --- |
| update Forlder set Parent\_ID = (select max(ID) from Folder where Name = ‘Folder2’ and TRUE\_FOLDER = True) where id = (select max(ID) from Folder where Name = ‘Folder1’ and TRUE\_FOLDER = True)  -- for the condition that the Name attribute for object folder field is unique |

To find information for an "Item" object that contains the text "item" in its name:

|  |
| --- |
| select \* from Folder where upper(name) like ‘%ITEM%’ and TRUE\_FOLDER = False |

To search for information in the hierarchy, use SQL with “**select … from … start with … connect by .. prior …**”