**Description of the problem statement:**

The title for the assignment was to create Instrument Management Application. The requirements were to -

* create a user interface which allow users to add, update and delete instruments.
* Use XML file for storing instruments data.
* User interface should have a layout which can show instrument data already present in XML file.
* Instrument data needs to be inputted as textbox for Name, combo box for User, list box for projects where multiple selections should be allowed.

There were other requirements of the assignment such as -

* There should be an interface declaration assembly where methods for add, update, delete instruments are declared only. Further I would need to use this interface to create objects dynamically as well.
* There should be an assembly where methods of interfaces should be defined. These methods further should be used in user interface part.
* User interface should have only display part.

**Overall description of the assignment:**

Among the weekly tasks, first task was to create the interface DLL. So, I created a class library in visual studio. There I had to create an interface in which I declared the methods for showing, adding, updating, and deleting the instruments. The methods declared were as –

1. public List<Instruments> getInstruments();
2. public void addInstrument(Instruments newInstrument);
3. public void deleteInstrument(string instrumentName);
4. public void updateInstrument(string toUpdateInstr, Instruments instr);

In the same DLL I created a class Instrument which contains all the properties that an instrument has such as –

1. public string Name { get; set; }
2. public string User { get; set; }
3. public string Project { get; set; }

After this, I set the output build path of this DLL to a folder and set the solution configurations to Release. This helps to make a DLL available in the specified folder when other file needs it.

Next task in the same week was to create another DLL, reader-writer which uses interface DLL and defines the methods of interface DLL. So, I created another class library in which there was only one class needed. In this DLL I added reference of the DLL from the folder mentioned earlier. Next, I implemented the interface in the class and provided body to the interface methods.

1. First method was public List<Instruments> getInstruments(). So, for this method I used a LINQ query which returns all records that are there in specified file given by path. The data in the XML file is converted to a list and as we know the return type of getInstruments method is List<Instruments>, the list is stored in a List<Instruments>. If there is no such file on the specified path, the method will return null.
2. Next method is public void addInstrument(Instruments newInstrument). This method takes an object of Instruments class as parameter. So, we’ll need to store properties of that object in some variables such as -

string \_name = newInstrument.Name;

string \_user = newInstrument.User;

string \_project = newInstrument.Project;

Further with the help of FileStream class, I accessed the file via path provided. Then loaded the xml file with the help of XmlDocument class. Then with the help of XmlElement class we created parent node named Instrument and next we created three child nodes for the parent node, each node was filled with respective instrument object property. At last, all the three child nodes are added to parent node. Next up we append the record just added, set file access of xml file to write, save the xml file and close it.

In case the provided path for xml file does not exist then a respective xml file will be generated dynamically, further the process of insertion is same as above.

1. Next method is public void deleteInstrument(string instrumentName). As we can see this method takes a string as parameter. This instrumentName is nothing but a property of instruments class object. So, we’ll load the xml file first then we’ll traverse through every Instrument parent node. Further it checks if Name node’s value is equal to parameter of the method. When the match is found then that Name node’s parent node is deleted from xml file, then save the file and break the traversing loop.
2. Next method was public void updateInstrument(string instrName, Instruments instrument). This method has two parameters first one is a string, and the second parameter is object of class Instruments. This method searches the desired node by instrName, traverses the xml file same way as done in delete method. Once the respective node is found then node specific value is set, in following way –

item.Element("Name").SetValue(instrument.Name); item.Element("User").SetValue(instrument.User); item.Element("Project").SetValue(instrument.Project);

Before I create any user interface, my task was to create a xml file. Because last task of week one was to check if CRUD operations take place on xml file. I created a xml file InstrumentsDB.xml which I used as database file. The xml file should have some schema, so I created a xsd file for the respective xml file. The XML schema definition (xsd) is as follows -

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="Instruments">

<xs:complexType>

<xs:sequence>

<xs:element name="Instrument" maxOccurs="unbounded" minOccurs="0">

<xs:complexType>

<xs:sequence>

<xs:element type="xs:string" name="Name"/>

<xs:element type="xs:string" name="User"/>

<xs:element type="xs:string" name="Project"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

Then for week 2 my task was to create user interface for the application. For displaying data, I used DataGrid layout in wpf. Below that I put 4 buttons as load data, delete instrument, add instrument, update instrument to perform their respective functionalities. For add instrument and update instrument I was told to create separate dialog boxes. I used MVVM for most of the functionalities in the user interface. Next task was to integrate interface DLL and reader-writer DLL in the user interface project. Another task in week 2 was to create an object of interface while creating reader-writer class dynamically using assembly name and class name. Before successful build I changed the output path to same folder as earlier both DLL and set the solution configurations to Release. So, it makes a folder which can work independently as a normal application. By the beginning of week 3, the application was working achieving the end-to-end functionality.

**Learnings from the assignment:**

1. Creating and using independent DLLs:

* In this assignment I’ve created two different DLLs and made use of them effectively.
* The main use of class libraries is that they are reusable and hence reduces the lines of code.

1. Code organization:

* We should split the code in different classes or projects.
* We should create necessary folders.
* We should place files in their respective folders(e.g., all viewmodels in ViewModel folder).

1. Code maintenance:

* We should write clean code without any unnecessary declarations.
* Place the declared variables, properties, and methods at the top.
* Remove all blank spaces, unnecessary comments.

1. MVVM:

* Model-View-ViewModel is an architecture widely used in WPF applications.
* In normal WPF applications, logic for WPF controls is written in code behind file only.
* But with the help of MVVM pattern in WPF, we can bind different properties to XAML code directly.
* This approach makes the WPF application loosely coupled.
* INotifyPropertyChanged interface is used to notify if there are any changes occurring in the control where the property is bonded.
* With the help of ICommand interface we can write commands which can be helpful with components like buttons.

1. Dynamic class loading using assembly name and class name:

* This feature allows us to create an object just by using an assembly name and class name of which we want to create an object.
* To use this feature, we don’t even need to add reference of the DLL where the class is present.

1. Using xml file as database:

* We can use xml files to store data with the help of child nodes.
* Basic requirement of an xml file is that it should have one root node.

**Sequence Diagram:**

Diagram

Description automatically generated

**Flowchart:**

Diagram

Description automatically generated

**Block diagram:**

Chart, diagram, box and whisker chart

Description automatically generated

**Conclusion:**

Hence, I have completed the PIP assignment Instrument Management System using MVVM architecture in WPF and XML file as database.