Programmed Solution to a Problem - Post Prototype Refinement

Porth-y-waen Silver Band Management System

Nia Hawkins: 7183
The Maelor School: 68146

Contents

Post prototype refinement

I have obtained feedback from a peer and a member of the band's committee. They both spent time looking at the prototype system and entering example data to test if the system works well and is intuitive. I will describe and analyse if the feedback will be implemented into the final program.

Automatic ID generation

In the prototype, the user can change the IDs. It has been suggested that users should not be able to do this as it could disrupt the file system and cause the wrong record to be changed or displayed. All ID text boxes will be changed to read-only so they cannot be changed.

It has also been highlighted in feedback that users will not be able to create the IDs for the inputs. The user could easily enter an ID already stored in the system or they could enter it in the wrong format. Setting the ID textboxes as read-only also creates the need for automatic IDs as the user will no longer be able to input them. To solve this issue, the system will have automatic ID creation, by searching for the highest ID number in the corresponding file and adding 1 to produce a new, unique ID. The pseudocode is shown below.

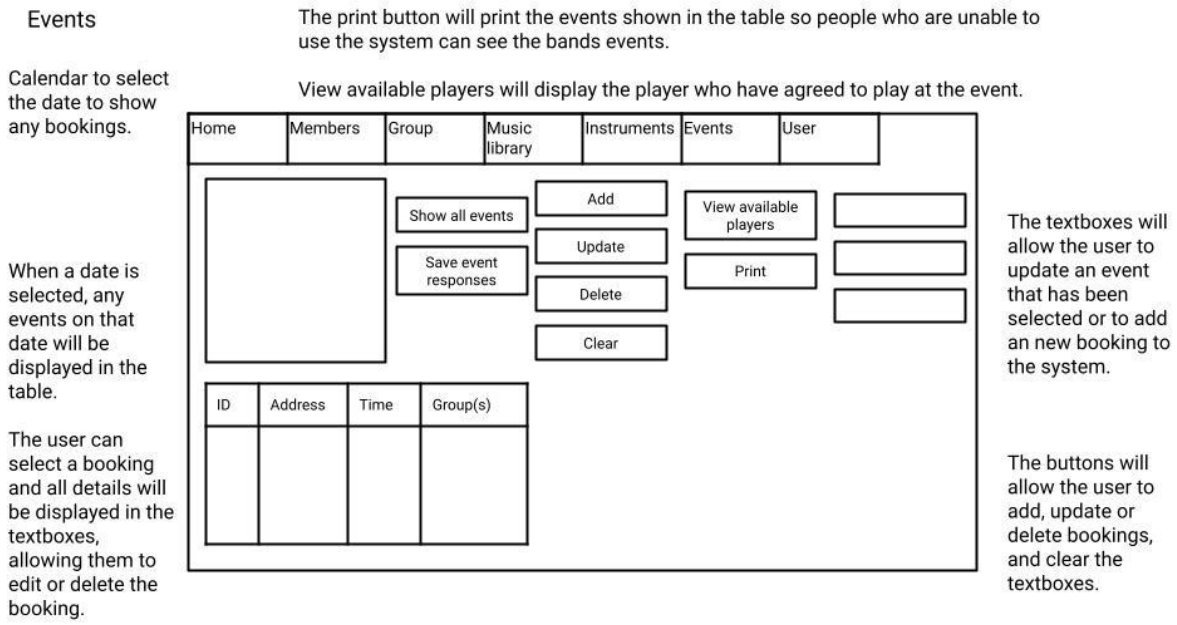
```
index is Integer
OPEN FILE “filename.txt”
totalRecords is Integer = LOF(1) / Len(filename.txt)
searchID is Integer = "00001”
IF totalRecords = 0 THEN
        txtID.Text = "00001"
END IF

FOR index = 1 To totalRecords
        READ FILE “filename.txt”
        // search for available id
        IF searchID = readId THEN
                searchID += 1
        END IF
NEXT
idString is String = searchID.ToString()
idLen is Integer = Len(idString)
finalID is String = idString
finalID = idString.PadLeft(5, "0")
txtID.Text = finalID
```

This requires the program to search a file for the first available ID and if it is found, the possible ID is incremented. This feature will reduce the chance of any error in the program and ensures that all the primary keys in each data structure table are unique.

Print events

As not all band members can use technology as well as others, it has been suggested that the user could be able to save and print the events bookings. I decided to add this feature to the system as it will still allow those who cannot use a computer or chose not to use the system to still get event details while making it easier for someone to give them the details.

An improved design including the buttons for printing is shown below.



To do this, the DataGridView is stored as a bitmap, which is then displayed in a print preview, allowing the user to print the bookings. This method also allows the user to choose what to show, using the show all button, or selecting a date in the calendar. The pseudocode for this is shown below.

```
Sub btnPrint_Click()
        PrintDocument.DefaultPageSettings.Landscape = True
        PrintPreviewDialog.ShowDialog()
End Sub

Sub PrintDocument_PrintPage()
    DECLARE bm is New Bitmap(700, dgvDay.Height)
    dgvDay.DrawToBitmap(bm, New Rectangle(30, 30, 700, dgvDay.Height))
    e.Graphics.DrawImage(bm, 0, 0)
End Sub
```

This requires extra data to be stored as a bitmap is created of the DataGridView which is printed.

View available players for an event

It was suggested that committee members should be able to view who can attend an event in the system. This will aid the management of events and will make it easier to find deps to fill any gaps. This requires further data to be stored. The data structure table is shown below for this file.

Event responses

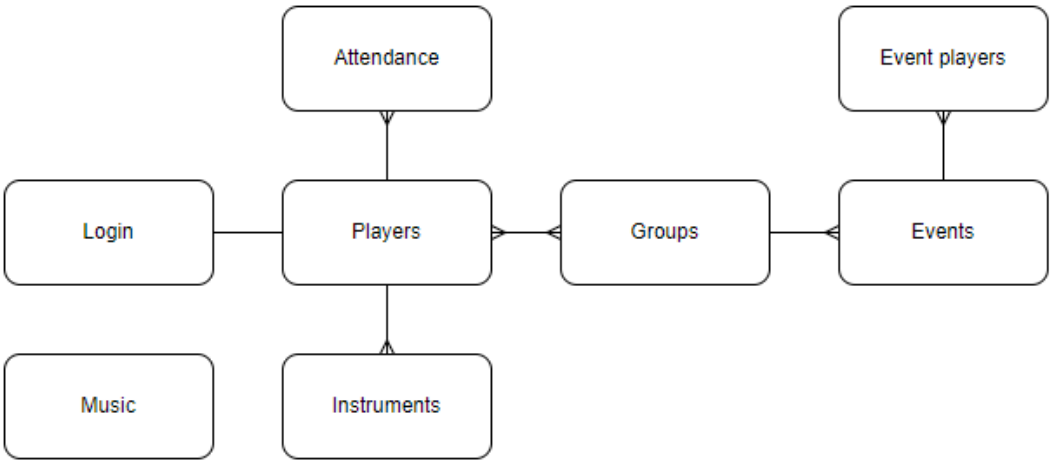
Field Name	Data Type	Description	Length	Example	Validation
responseID *PK	string	The unique identification number of the response	5	00382	**Length check** - must be 5 characters long.
players	string	The IDs of the players that are playing for the event	400	00001, 00003, 00012	**Type check** - data must be entered as a boolean through the dataGridView.

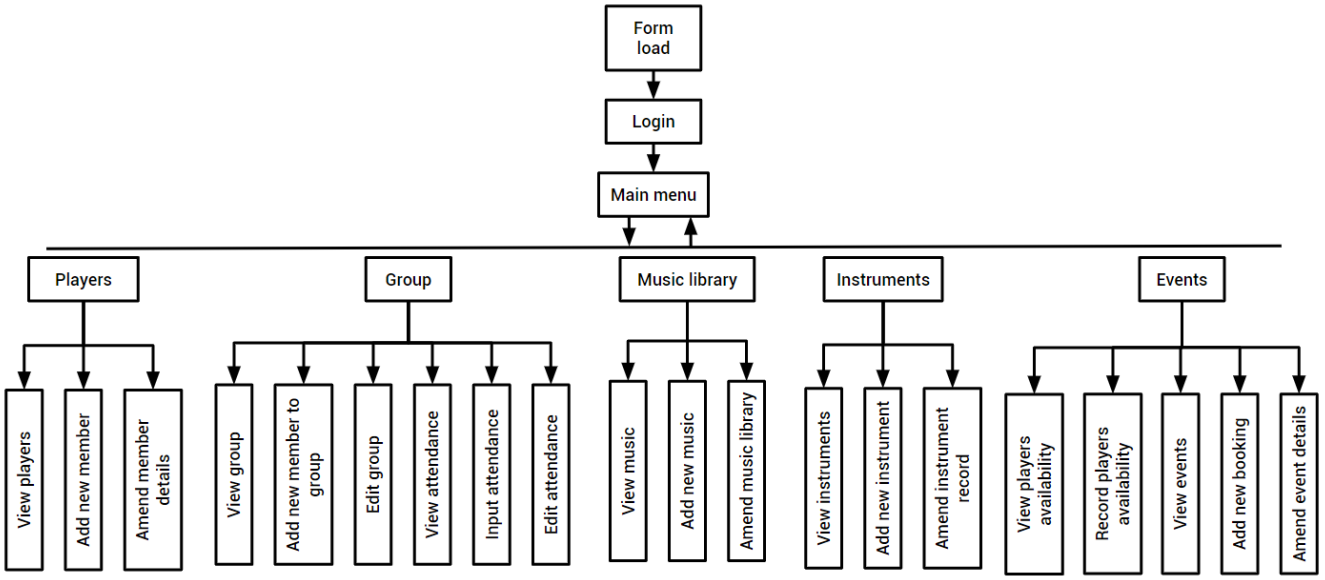
The user can check next to each event if they can attend or not, and then click a button to save their response. The pseudocode for this is shown below.

```
index is Integer
oneResponse is responses 'pointer
OPEN FILE(1, "eventsResponses.dat", OpenMode.Random,,, Len(oneResponse))

 totalRecords is Integer = LOF(1) / Len(oneResponse)
checked is Boolean
added is Boolean

// store the response
IF totalRecords = 0 THEN
        // get response from dgv
        checked = dgvDay.Rows(index).Cells(6).Value

        // only store if true
        IF checked = TRUETHEN
                oneResponse.responseID = dgvDay.Rows(index).Cells(0).Value
                oneResponse.playing = login.userID & ", "   //append to responses
                WRITE FILE(1, oneResponse, totalRecords + 1)
        END IF
END IF

FOR index = 0 To dgvDay.Rows.Count - 1
        added = FALSE
```

```
        FOR j = 0 To totalRecords
                READ FILE(1, oneResponse)

                // store the response
                IF oneResponse.responseID = dgvDay.Rows(index).Cells(0).Value THEN
                        // get response from dgv
                        checked = dgvDay.Rows(index).Cells(6).Value

                        // only store if true
                        IF checked = TRUE THEN
                                // append to existing responses
                                oneResponse.playing = oneResponse.playing & login.userID & ", "
                                WRITE FILE(1, oneResponse, totalRecords + 1)
                                added = TRUE
                        END IF
                END IF
        NEXT

        If added = False Then
                // store the response
                // get response from dgv
                checked = dgvDay.Rows(index).Cells(6).Value

                // only store if true
                IF checked = TRUE THEN
                        // append to responses
                        oneResponse.playing = oneResponse.playing & login.userID & ", "
                                WRITE FILE(1, oneResponse, totalRecords + 1)
                END IF
        END IF
NEXT
CLOSE FILE(1)
```

As a new file is stored, the Entity Relationship Diagram and System Overview have been updated to show this change.



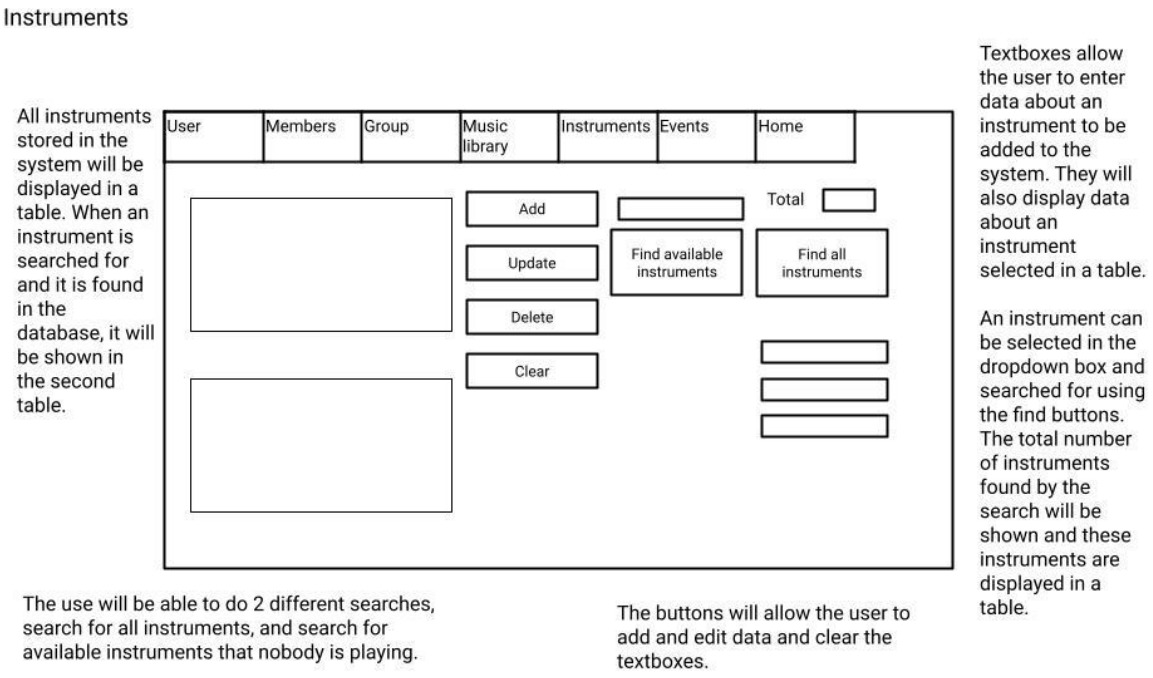


Processes
1. Form load - displays a welcome message to the user
2. Login - allows the user to enter the system
3. Main menu - displays all menu options to the user, allowing them to easily navigate the system
4. Players - allows user to view and edit player details
5. Group - allows user to view and edit group and mark attendance
6. Music library - allows user to view and edit music library
7. Instruments - allows user to view and edit the instrument record
8. Events - allows user to view and edit event bookings and player availability

Instrument searches

The committee member said that they did not know what *btnFind* did on the instruments screen and had thought it had failed when it did not show any instruments when there were instruments of the type selected. *btnFind* searches for available instruments so if the instruments had a holder ID, they would not be shown. In order to make the interface more intuitive, I changed the text on the buttons and added a new button to search for all instruments. This search uses the same code as the existing instrument search, but will display any instruments found, if they have a holder ID stored or not. No extra data is required.

The improved designs for this screen are shown below.



Undo and redo

The committee member suggested that the screens have undo and redo buttons to aid the input of data. This would make the interface more intuitive and user-friendly, and could make it quicker for the user to enter data.

The following pseudocode shows a possible method of implementing an undo button on the players screen for some of the inputs.

```
// declaring global variables to handle undo
        DECLARE Undo is array
        DECLARE count is Integer = 0
        DECLARE pointer is Integer = 1
```

```
Sub formChanged
        // saves the data in the form to a dynamic array so changes can be restored
        DECLARE changeRecorded is Boolean = FALSE
        DECLARE i is Integer
        IF count = 0 THEN
                i = 0
                FOR i = (count * 14) To undo.Length
                        IF changeRecorded = FALSE THEN
                                ReDim Preserve undo(UBound(undo) + 14)
                                IF undo(i) = Nothing And i > 0 THEN
                                // search for empty place in array
                                        undo(i) = txtName.Text
                                        undo(i + 1) = dtpDOB.Text
                                        undo(i + 2) = txtEmail.Text
                                        undo(i + 3) = txtPhone.Text

                                        pointer += 4
                //save the number of times data is stored in the array so start location can be found
                                        count += 1
                                        changeRecorded = TRUE
                                END IF
                        END IF
                NEXT
        END IF
END Sub

// restores previous outputs when undo clicked
Sub undo
        DECLARE startLocation is Integer = pointer - 9
        IF count = 1 THEN  // if only one change has been saved, the form must have been blank
previously
                txtName.Clear()
                dtpDOB.ResetText()
                txtPhone.Clear()
                cmbInstrument.ResetText()
                cmbLevel.ResetText()
                chkPhotoPerm.CheckState = False
                chkPSB.CheckState = False
                chkPYTB.CheckState = False
                chkPBB.CheckState = False
                chkStarters.CheckState = False
                cmbRole.ResetText()
                txtContName.Clear()
                txtContPhone.Clear()
```

```
                FOR j = 1 To 14
                        undo(j) = ""
                NEXT

                count = 0
                pointer = 0

                ELSEIF count = 0 THEN
                        OUTPUT "No changes made to be undone”

                ELSEIF count = 2 THEN
                        txtName.Text = undo(1)
                        dtpDOB.Text = undo(2)
                        txtEmail.Text = undo(3)
                        txtPhone.Text = undo(4)

                        FOR j = 1 To 8
                                undo(j) = ""
                        NEXT

                        count = count - 1
                        pointer = pointer - 8

                ELSE    // if more than 2 changes
                        txtName.Text = undo(startLocation)
                        dtpDOB.Text = undo(startLocation + 1)
                        txtEmail.Text = undo(startLocation + 2)
                        txtPhone.Text = undo(startLocation + 3)

                        FOR j = startLocation To startLocation + 8
                                undo(j) = ""
                        NEXT
                        count = count - 1
                        pointer = pointer - 8
                END IF
END Sub
```

This algorithm did not always work correctly, as sometimes during repeated undos, some data in the array would be skipped, and the data inputted two changes previously would be outputted instead of the expected data. Another difficulty was implementing a redo button. When testing, this had a similar algorithm to the undo button. This also did not always work as expected in the same way as the undo button. When testing an undo and then redo, further issues occurred and the program would often crash.

After considering and researching other ways to implement this feature, I decided not to implement this feature, due to its complexity and the limited timeframe to complete the project, as I did not have the time to spend focusing on a small feature not included in the success criteria and that did not hugely affect the data processing.

Instrument service dates

The committee member also said that it can be difficult to know if an instrument needs servicing so it would be useful if it was indicated next to the instruments when they are displayed. I chose to change the colours of the dates in the DataGridView to intuitively show if they are due servicing. I decided this did not need to be data stored in a file as it depends on the current date. If it has been 11 months or more since the last service, the date will be red, 10 months since it will be orange, yellow if it has been 10 months, and 9 months or earlier, it will be green.

The following pseudocode shows how data is displayed and formatted on the instruments screen.

```
Sub colours()
 // formats and displays data in dgv
        // define colour variables to format service date colours
        DECLARE red is date= Date.Today.AddMonths(-11)
        DECLARE yellow  is date= Date.Today.AddMonths(-9)
        DECLARE orange  is date= Date.Today.AddMonths(-10)

        DECLARE serviceDate is String
        DECLARE index is Integer
        DECLARE oneInstrument is instruments              // pointer to structure
        DECLARE oneMember is memberInfo    // called to get holderName from id
        DECLARE totalRecordsMember is Integer
        DECLARE totalRecordsInstrument is Integer

        dgvInstruments.Rows.Clear() // clear rows in datagridview

        // open file and get total number of records in the file
        OPEN FILE(1, "instruments.dat", OpenMode.Random,,, Len(oneInstrument))
        totalRecordsInstrument = LOF(1) / Len(oneInstrument)

        // if no records in the file, output a message
        IF totalRecordsInstrument = 0 THEN
                OUTPUT "No instruments stored"
        END IF

        // open file and display each record in dgv
        FOR index = 1 To totalRecordsInstrument
                OPEN FILE(2, "players.dat", OpenMode.Random,,, Len(oneMember))
```

```
                totalRecordsMember = LOF(2) / Len(oneMember)
                READ FILE(1, oneInstrument)

                // if holderID is empty, display the record without holderName
                IF oneInstrument.holderID = empty THEN
                        dgvInstruments.Rows.Add(oneInstrument.instrumentID,
oneInstrument.serialNumber.Trim(), oneInstrument.name.Trim(), oneInstrument.instrument,
oneInstrument.holderID.Trim(), "", oneInstrument.serviceDate)
                // if holderID is not empty, find holderName in players file and output it in dgv
                ELSE
                        FOR i = 1 To totalRecordsMember // find holderName
                                READ FILE(2, oneMember)
                                IF oneMember.id.Contains(oneInstrument.holderID.Trim()) THEN
                                        dgvInstruments.Rows.Add(oneInstrument.instrumentID.Trim
                                        (), oneInstrument.serialNumber.Trim(),
                                        oneInstrument.name.Trim(),
                                        oneInstrument.instrument.Trim(),
                                        oneInstrument.holderID.Trim(), oneMember.name.Trim(),
                                        oneInstrument.serviceDate)
                                        EXIT FOR
                                END IF
                        NEXT
                END IF
                CLOSE FILE(2)
        NEXT
        CLOSE FILE(1)

        // format the service date colours in each row in dgv
        FOR currentRow = 1 To dgvInstruments.RowCount
                serviceDate = dgvInstruments.Rows(currentRow - 1).Cells(6).Value

                // if more than 11 months ago, cell is red
                IF serviceDate <= red THEN
                        dgvInstruments.Rows(currentRow - 1).Cells(6).Style.BackColor =
                Color.DarkRed
                        dgvInstruments.Rows(currentRow - 1).Cells(6).Style.ForeColor = Color.White

                 // if 10 months ago, cell is orange
                ELSEIF serviceDate <= orange And serviceDate > red THEN
                        dgvInstruments.Rows(currentRow - 1).Cells(6).Style.BackColor =
        Color.DarkOrange
                        dgvInstruments.Rows(currentRow - 1).Cells(6).Style.ForeColor = Color.Black

                  // if 9 months ago, cell is yellow
```

```
                ELSEIF serviceDate <= yellow And serviceDate > orange THEN
                        dgvInstruments.Rows(currentRow - 1).Cells(6).Style.BackColor = Color.Gold
                        dgvInstruments.Rows(currentRow - 1).Cells(6).Style.ForeColor = Color.Black



                // if less than 9 months ago, cell is green
                ELSEIF serviceDate > yellow THEN
                        dgvInstruments.Rows(currentRow - 1).Cells(6).Style.BackColor =
                Color.LightGreen
                        dgvInstruments.Rows(currentRow - 1).Cells(6).Style.ForeColor = Color.Black
                END IF
        NEXT
END Sub
```

The added colour formatting requires extra data to be collected, processed and stored. The program has to find the current date, which is used to calculate the dates that would result in different colours. These dates are stored in the colour variables.