Programmed Solution to a Problem - Design

Porth-y-waen Silver Band Management System

Nia Hawkins: 7183
The Maelor School: 68146

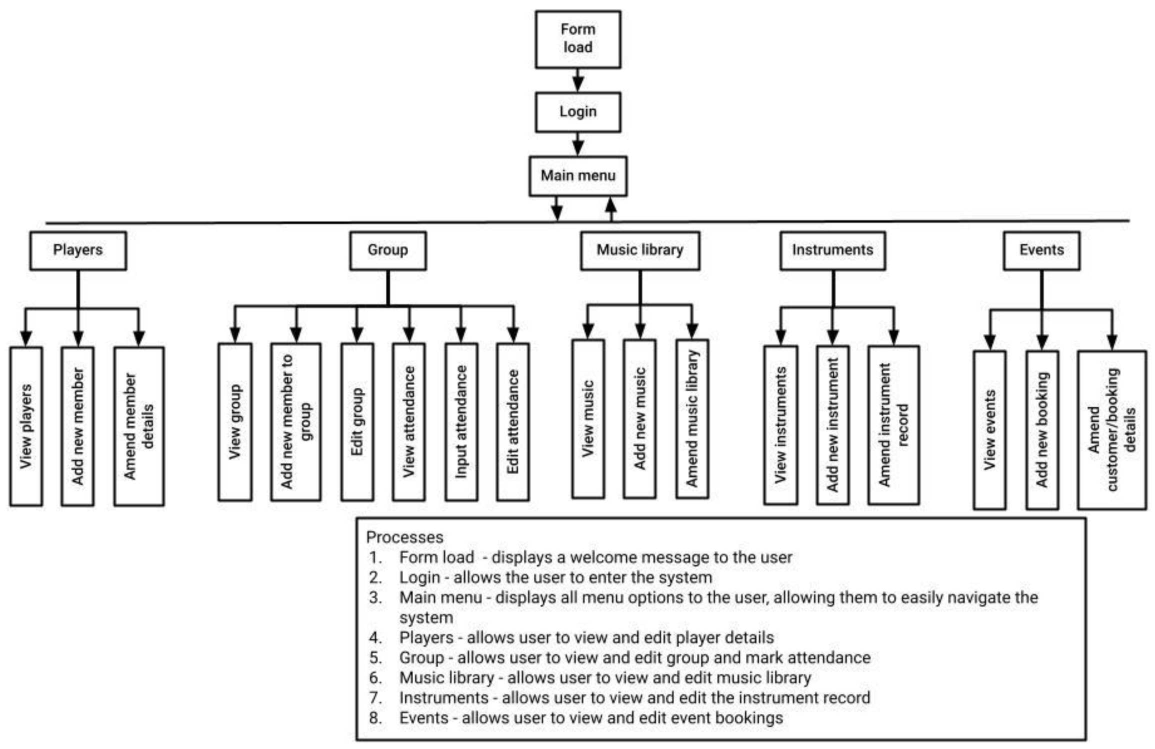
Contents

Design

After devising my objectives, I have created a series of designs to show the user interface, data structures and processing stages.

System Overview

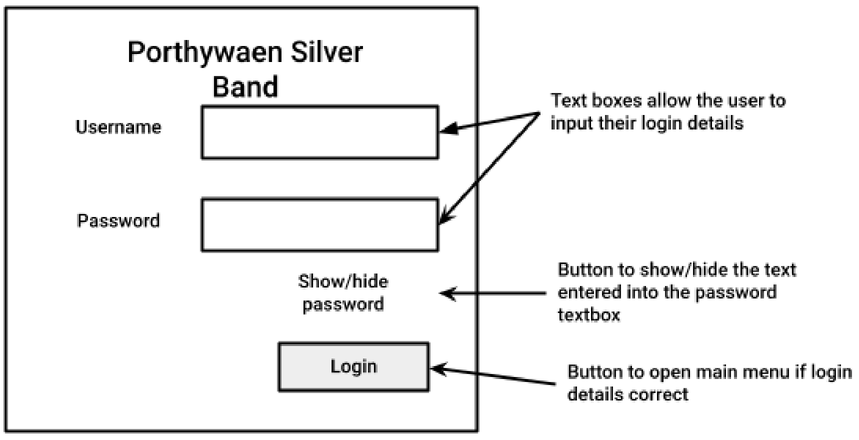
I have produced an overview of the system to show how different windows will link together in the final system.



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

Forms

Login

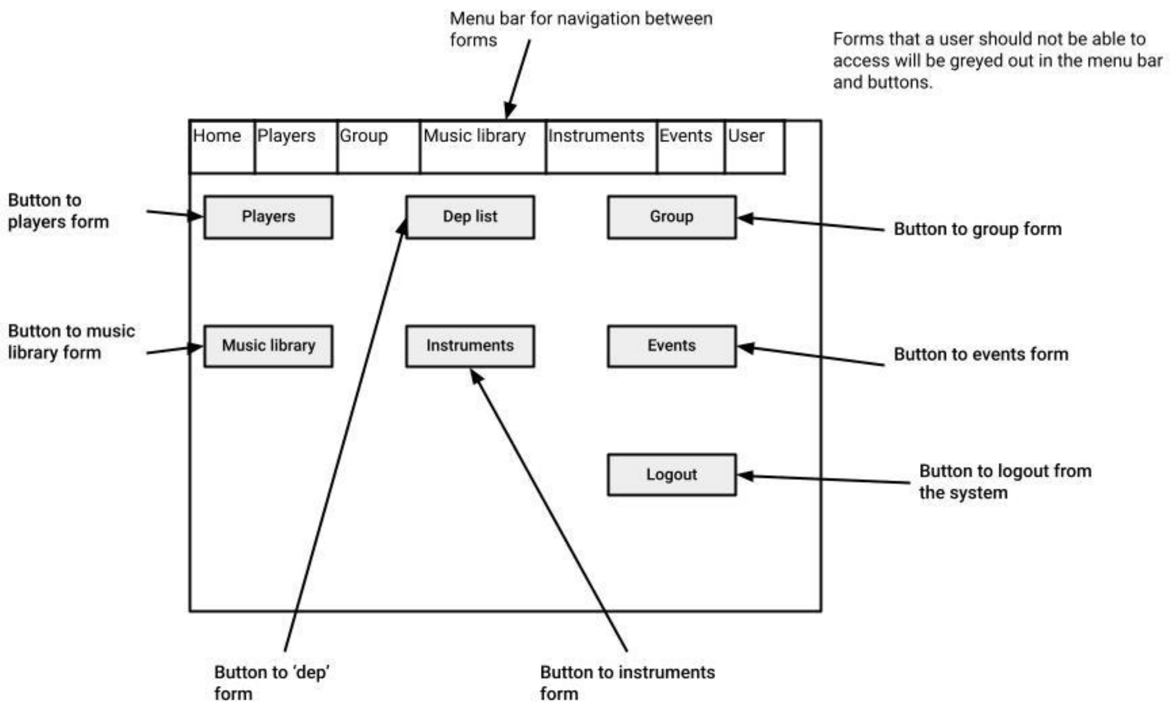

Porthywaen Silver Band
Username
Password
Show/hide password
Login
Text boxes allow the user to input their login details
Button to show/hide the text entered into the password textbox
Button to open main menu if login details correct

This is the login page for the system. It allows the user to enter their username and password that is given to them when they become a member. All usernames and passwords will be encrypted to ensure data is kept secure.

The “reset password” button allows the user to change their password. This will ask the user to enter their username and it will allow them to change their password.

When the user clicks the “login” button, the system will check if the username and password match those stored in the database. If it is correct, the main menu will be displayed. If it is incorrect, an error message will be displayed, prompting the user to try again.

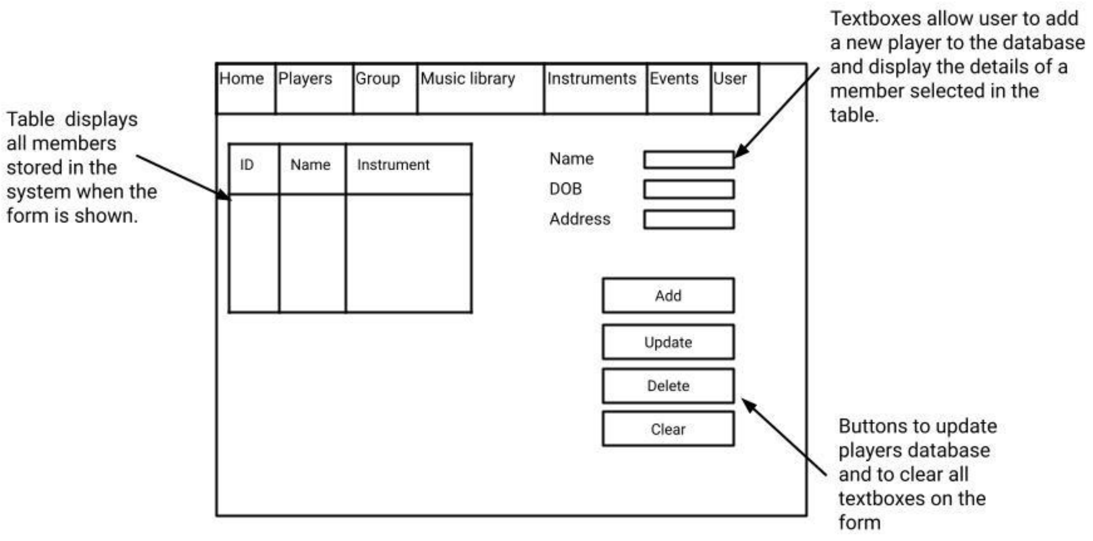
Home


Menu bar for navigation between forms
Forms that a user should not be able to access will be greyed out in the menu bar and buttons.
Home
Players
Group
Music library
Instruments
Events
User
Button to players form
Players
Dep list
Group
Button to group form
Button to music library form
Music library
Instruments
Events
Button to events form
Logout
Button to logout from the system
Button to 'dep' form
Button to instruments form

This screen will give the user intuitive access to all features of the system. When a button is clicked, the selected screen will be displayed. Each user will have different levels of access, based on their status within the band. This means that not all members need to have access to all areas of the system. This will be done by disabling the buttons on the home screen and the links in the menu bar.

All screens in the program have the same menu bar to make it easier for the end user to learn how to use the system.

Players

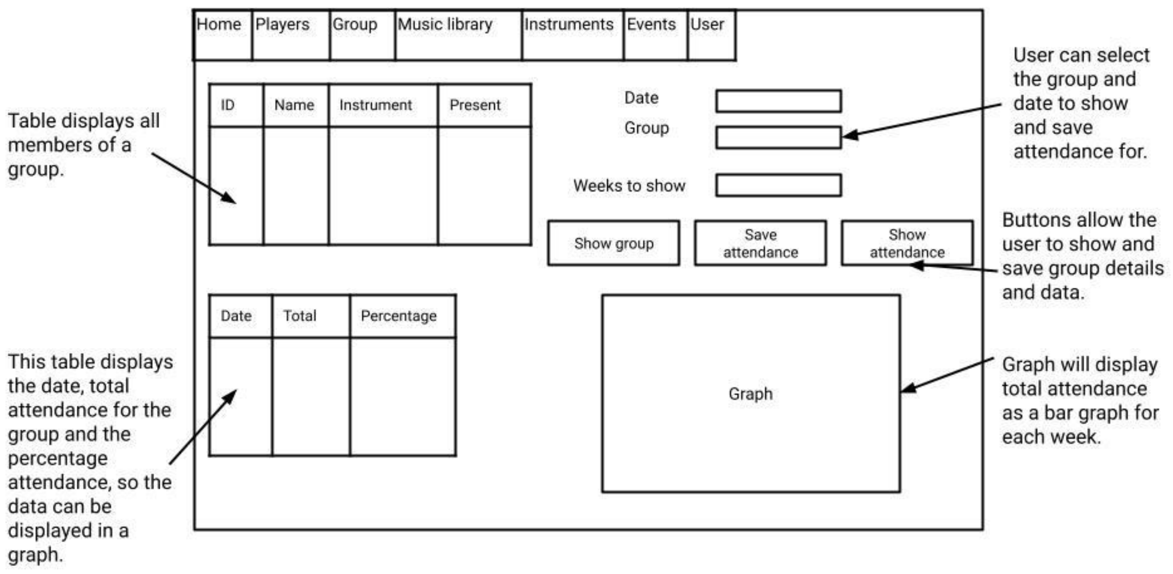

Textboxes allow user to add a new player to the database and display the details of a member selected in the table.
Home
Players
Group
Music library
Instruments
Events
User
Table displays all members stored in the system when the form is shown.
ID
Name
Instrument
Name
DOB
Address
Add
Update
Delete
Clear
Buttons to update players database and to clear all textboxes on the form

This screen will allow the user to view and manage all players in the system. The user will be able to add a new player to the system by entering the data into the input boxes. These will include text boxes for most inputs, but will also have combo boxes, checkboxes and a calendar to make the input more intuitive for the user and to reduce the chance of errors.

The table will show all players that are in the system. The table will be sorted by clicking each column heading to sort by the selected field. When the user clicks a player in the table, all of their details will be displayed in the textboxes and selectors, allowing all details to be viewed and edited. The player will also be deleted from the system after they have been selected. The textboxes also allow the user to add a new player to the system.

Buttons on the screen will allow the user to clear the boxes so a new player will be added, and to add, update or delete a player.

Group

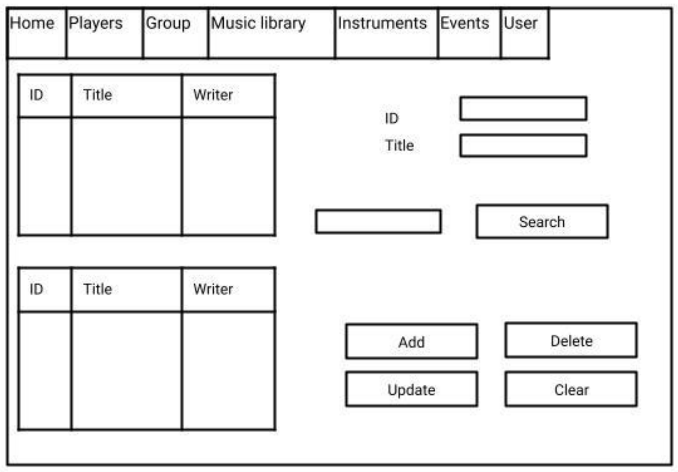

Home
Players
Group
Music library
Instruments
Events
User
ID
Name
Instrument
Present
Date
Group
Weeks to show
Show group
Save attendance
Show attendance
Date
Total
Percentage
Graph
Table displays all members of a group.
This table displays the date, total attendance for the group and the percentage attendance, so the data can be displayed in a graph.
User can select the group and date to show and save attendance for.
Buttons allow the user to show and save group details and data.
Graph will display total attendance as a bar graph for each week.

The user will be able to view and manage members of a group and their attendance using this screen.
The user will select a date in a calendar and a group in a combo box, and all members of the group will be displayed in a table when the *show group* button is clicked. In the table, the Present column will contain checkboxes so the user will mark players' attendance for the selected date. The attendance for the selected date will be saved by clicking the *save attendance* button.

When the *show attendance* button is clicked, the program will calculate the percentage attendance of the group for the total number of dates entered in the *weeks to show* textbox. This data will then be displayed as a bar graph.

Music Library

All music stored in the system will be displayed in a table. When a piece is searched for and it is found in the database, it will be shown in the second table.



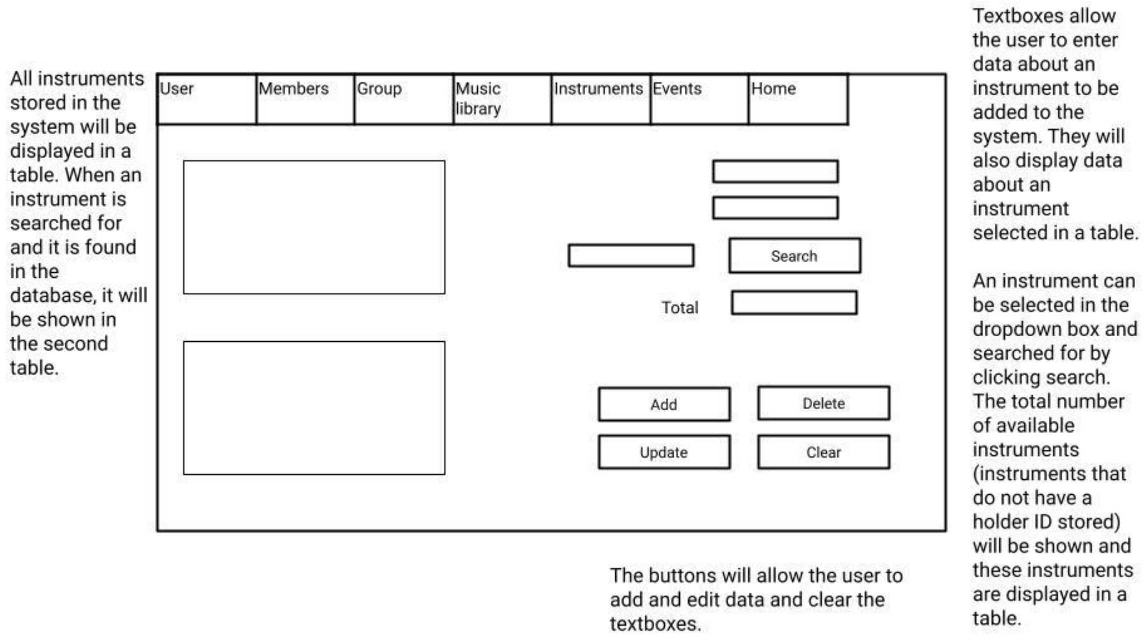
The textboxes allow data to be inputted and a selected piace to be displayed.

The user can enter a title and click search to search the database for the piece.

The buttons allow the user to add and edit data and clear all the inputs.

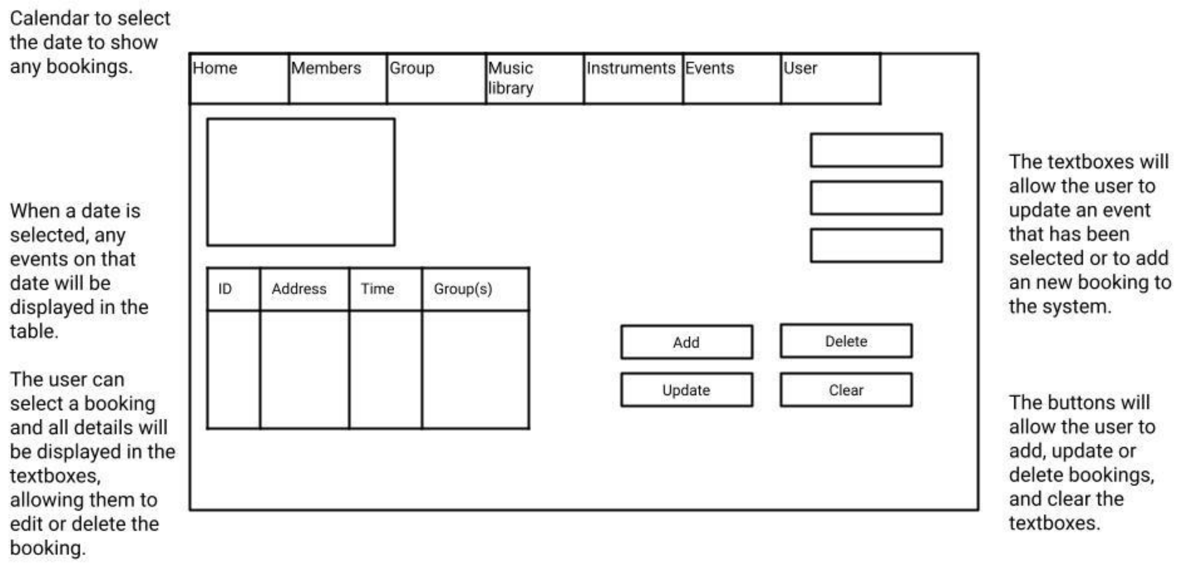
The table headers can be clicked to sort the data and a piece in the table can be clicked to display all the details stored about it in the textboxes. The user can also enter a new piece into the system using the text boxes. The system can be searched for a piece of music by entering the title and clicking search. If the music is found, it will be displayed in the second table. The user will be able to update a piece's details or delete the music by selecting it in the table, editing the details if they need to be changed and then clicking the relevant button.

Instruments


All instruments stored in the system will be displayed in a table. When an instrument is searched for and it is found in the database, it will be shown in the second table.
User
Members
Group
Music library
Instruments
Events
Home
Search
Total
Add
Delete
Update
Clear
Textboxes allow the user to enter data about an instrument to be added to the system. They will also display data about an instrument selected in a table.
An instrument can be selected in the dropdown box and searched for by clicking search. The total number of available instruments (instruments that do not have a holder ID stored) will be shown and these instruments are displayed in a table.
The buttons will allow the user to add and edit data and clear the textboxes.

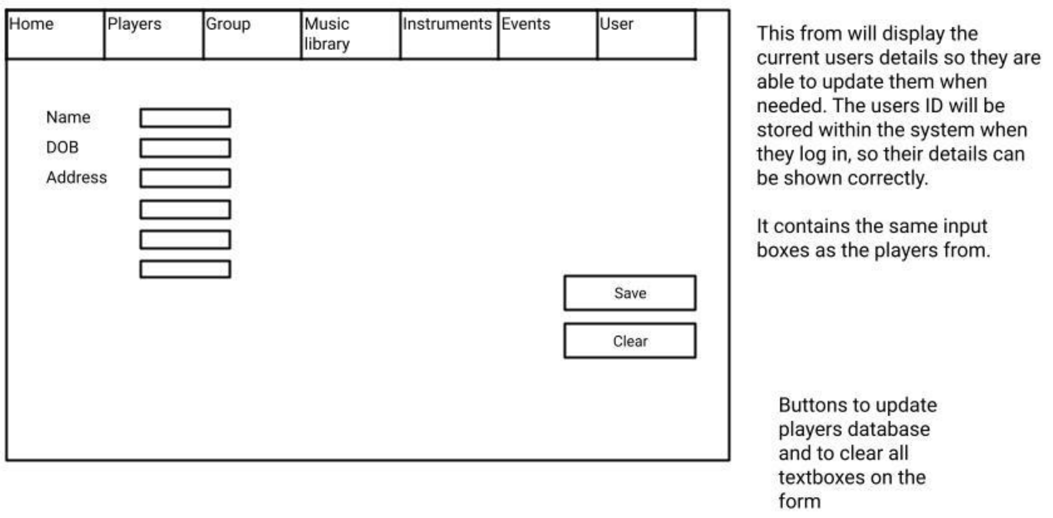
The instruments in the system will be displayed in the first table. The table headers can be clicked to sort the data and a piece in the table can be clicked to display all the details stored about it in the textboxes. This allows the user to update or remove an instrument from the system. The user will be able to search for available instruments by selecting the instrument in the dropdown box and clicking search. Any available instruments will be shown in the second table.

Events


Calendar to select the date to show any bookings.
When a date is selected, any events on that date will be displayed in the table.
The user can select a booking and all details will be displayed in the textboxes, allowing them to edit or delete the booking.
Home
Members
Group
Music library
Instruments
Events
User
ID
Address
Time
Group(s)
Add
Delete
Update
Clear
The textboxes will allow the user to update an event that has been selected or to add an new booking to the system.
The buttons will allow the user to add, update or delete bookings, and clear the textboxes.

The user will be able to use a calendar to view event bookings. When they select a date in the calendar, any booking will be shown in the table. The user will be able to click the booking in the table, allowing them to view all the details so they can be changed, or the booking can be deleted. A new booking will be added to the system by selecting a date in the calendar and entering the details.

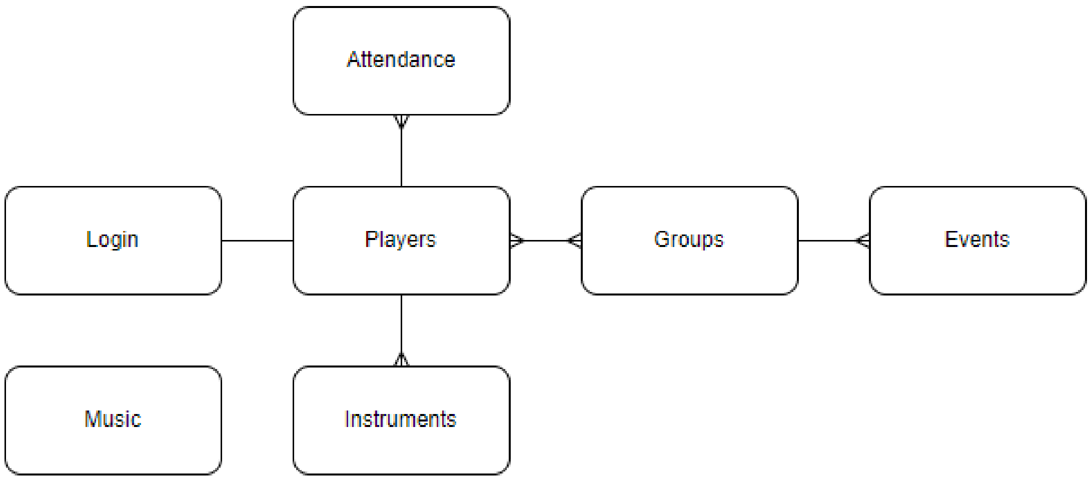
User


Home
Players
Group
Music library
Instruments
Events
User
Name
DOB
Address
Save
Clear
This from will display the current users details so they are able to update them when needed. The users ID will be stored within the system when they log in, so their details can be shown correctly.
It contains the same input boxes as the players from.
Buttons to update players database and to clear all textboxes on the form

The user will be able to view and edit their own details. They will not be able to change their ID as this is the primary key that identifies them in other files.

Data structures

Entity Relationship Diagram



The entity relationship diagram shows how each of the tables and files in the program link together.

Data structure tables

Player details

Field Name	Data Type	Description	Length	Example	Validation
ID *PK	String	A player's unique identification number	5	00024	**Length check** - must be 5 characters long.
Name	String	The name of the player	50	James Richards	**Presence check** - data must be entered.
DOB	Date	The date of birth of the player	24	10/1/1989	**Type check** - data must be a date. **Range check** - year must be between 1900 to the current date.
Email	String	The email address of the player	50	jamesrichards @gmail.com	**Format check** - must follow email format, and must contain "@" and ".",
Phone	String	The player's phone number	11	07964837615	**Length check** - must be exactly 11 characters long.
Instrument	String	The instrument the player plays	14	Cornet	**Lookup table** - shows a list of all possible instruments. **Presence check** - data must be entered.
Level	Integer	The playing level of the player	2	5	**Lookup table** - shows a list of all possible playing levels.. **Presence check** - data must be entered.
Role	String	The role of the player in the band's management	20	Player	**Lookup table** - shows a list of all possible playing levels. **Presence check** - data must be entered.

PhotoPerm	Boolean	If the player has given permission for photographs of them to be posted on social media	1	True	**Type check** - data must be boolean.
contName	String	The name of the player's emergency contact	50	Sophie Richards	**Presence check** - data must be entered.
contPhone	String	The player’s emergency contact phone number	11	07436488201	**Presence check** - data must be entered.
Groups	String	The groups the player is a member of.	24	PSB, PYTB	**Type check** - data must be boolean.

Group

Field Name	Data Type	Description	Length	Example	Validation
ID *PK	String	A player’s unique identification number	5	00024	**Length check** - must be 5 characters long.
Name	String	The name of the player	50	James Richards	**Presence check** - data must be entered.
Instrument	String	The instrument the player plays	14	Cornet	**Lookup table -** shows a list of all possible instruments **Presence check** - data must be entered.

Attendance

Field Name	Data Type	Description	Length	Example	Validation
ID *PK	String	A player's unique identification number	5	00024	**Length check** - must be 5 characters long.
Name	String	The name of the player	50	James Richards	**Presence check** - data must be entered.
markDate	String	The date the attendance has been recorded	22	12/04/2022	**Type check** - data entered must be a date **Range check** - year must be after 1900
Mark	String	If the player was present or not	5	True	**Type check** - data must be entered as boolean.
Group	String	The group the player is a member of.	8	PSB	**Presence check** - data must be entered. **Lookup table** - provides a list of acceptable inputs.

Music

Field Name	Data Type	Description	Length	Example	Validation
ID *PK	string	The unique identification number of the music	5	00294	**Length check** - must be 5 characters long.
title	String	The title of the music	50	In Flanders Fields	**Presence check** - data must be entered.
writer	String	The compost and/or arrange of the music	50	Gavin Somerset	**Presence check** - data must be entered.

Events

Field Name	Data Type	Description	Length	Example	Validation
eventID *PK	string	The unique identification number of the event	5	00382	**Length check** - must be 5 characters long.
responseID *FK	string	The unique identification number of the response	5	00382	**Length check** - must be 5 characters long.
customerID *FK	string	The unique identification number of the customer	5	00231	**Length check** - must be 5 characters long.
address	string	The address where the event will be held	100	Welshpool Town Hall, Welshpool	**Presence check** - data must be entered.
postcode	string	The postcode where the event will be held	8	SY21 7JQ	**Format check** - must be in the format LL99 9LL
eventDate	date	The date of the event	22	12/03/2022	**Type check** - data must be a date. **Range check** - must be after the current date
startTime	String	The time the event starts	15	3 pm	**Presence check** - data must be entered.
groups	string	The band's groups that will take part	24	PSB	**Type check** - data must be entered as a boolean.
music	string	The music the band will play	300	In Flanders Fields	**Presence check** - data must be entered.
arrivalTime	string	The time the players need to arrive at	10	2 pm	**Presence check** - data must be entered.
customerID *PK	string	The unique identification number of the music	5	00231	**Length check** - must be 5 characters long.

contName	string	The name of the event organiser	50	Tom Smith	**Presence check** - data must be entered.
contPhone	string	The organiser’s phone number	11	07847352841	**Length check** - must be 11 characters long.
contEmail	string	The organiser’s email address	50	tom.smith@gmail.com	**Format check** - must follow email format, and must contain “@” and “.”,

Instrument

Field Name	Data Type	Description	Length	Example	Validation
serialNumber *PK	string	The serial number of the instrument	10	AP84729JP8	**Presence check** - data must be entered.
name	String	The name of the instrument	30	Besson Sovereign	**Presence check** - data must be entered.
instrument	String	The type of instrument	14	Cornet	**Lookup table -** shows a list of all possible instruments **Presence check** - data must be entered.
holderID *FK	String	The unique identification number of the player that has the instrument	5	00033	**Length check** - must be 5 characters long, but only if data is present
serviceDate	Date	When the instrument was last serviced	22	23/07/2021	**Presence check** - data must be entered. **Type check** - data entered must be a date

Algorithms

Show data in DataGridView

The following pseudocode will be used to read all data from the relevant file and output each record in its own row in the DataGridView shown on the screen. It is used throughout the program where a DataGridView is used. When a screen is shown, or when the data in the file changes, this subroutine is automatically called to ensure the data that is outputted is correct.

The pseudocode below is used to output the data from “Players.dat” in dgvPlayers.

```
Sub dgvRefresh()
        DECLARE index is Integer
        DECLARE oneMember is memberInfo     // pointer to structure

        dgvPlayers.Rows.Clear()             // remove all rows from dgv
        OPEN FILE(1, "players.dat", OpenMode.Random,,, Len(oneMember))

        DECLARE totalRecords Is Integer = LOF(1) / Len(oneMember)

// read each record from file and add to row in dgv
        FOR index = 1 To totalRecords
                READ FILE(1, oneMember)
                dgvPlayers.Rows.Add(oneMember.id.Trim(), oneMember.name.Trim(),
oneMember.instrument.Trim(), oneMember.phone.Trim())
        NEXT
        CLOSE FILE(1)
END Sub
```

Add record

This algorithm will add any data to a file. It is used to add a player, instrument, music or event to the system. Once the record has been added successfully, the table which displays the data will refresh to display the new record.

The pseudocode below is used to add a new player to the “players.dat” file.

```
Sub Add()
    DECLARE oneMember is memberInfo    // pointer to structure
    DECLARE index is Integer
    OPEN FILE(1, "players.dat", OpenMode.Random,,, Len(oneMember))

    // populate structure
    oneMember.id = txtID.Text
    oneMember.name = txtName.Text
    oneMember.dob = dtpDOB.Text
    oneMember.email = txtEmail.Text
    oneMember.phone = txtPhone.Text
    oneMember.instrument = cmbInstrument.Text
    oneMember.level = cmbLevel.Text
    oneMember.photoPerm = chkPhotoPerm.Checked
    oneMember.contName = txtContName.Text
    oneMember.contPhone = txtContPhone.Text
    oneMember.groups = groups
    oneMember.role = cmbRole.Text
    oneMember.password = txtID.Text 'when member first added set password as id

    FILE WRITE(1, oneMember, totalRecords + 1)    // +1 append to file
    CLOSE FILE(1)
    OUTPUT "Player added"
    dgvRefresh()    // call subroutine that outputs file in dgv
END Sub
```

Delete record

This algorithm is used to delete a record from a file. It will remove a player, instrument, piece of music or an event booking from the system. Once the record has been added successfully, the table which displays the data will automatically refresh to remove the record.

The pseudocode below is used to delete a player from the “players.dat” file.

```
Sub Delete()
    DECLARE oneMember is memberInfo    // pointer to structure

    OPEN FILE(1, "players.dat", OpenMode.Random,,, Len(oneMember))
    OPEN FILE(2, "tempPlayers.dat", OpenMode.Random,,, Len(oneMember))

    DO WHILE NOT EOF(1)
        'if place in file isn't the record number of the record to be deleted, add the record to
the temp file
        IF Loc(1) <> currentRecord - 1 THEN
            READ FILE(1, oneMember)
            WRITE FILE(2, oneMember)
        ELSE
        'if it is the record number of the file to be deleted, skip the record and don’t write it
to temp file
            FILE READ(1, oneMember)
        End If
    LOOP
    CLOSE FILE(1)
    CLOSE FILE(2)

    DELETE("players.dat")
    RENAME("tempPlayers.dat", "players.dat")
    OUTPUT "Player deleted"

    dgvRefresh()    // call subroutine that outputs file in dgv
End Sub
```

Update record

This algorithm allows any data stored into the system to be updated. It will be used to update players, instruments, music and event bookings. The algorithm determines which record to update by storing the table row selected. Once the record has been added successfully, the table which displays the data will automatically refresh to display the changes.

The pseudocode below is used to update a player in the “players.dat” file.

```
Sub update
        DECLARE row is DataGridViewRow = dgvPlayers.CurrentRow
        DECLARE currentRecord is integer  = row.Index + 1   // save the record as the row selected
        DECLARE oneMember is memberInfo     // pointer to structure

        // storing inputs in structure
        oneMember.id = txtID.Text
        oneMember.name = txtName.Text
        oneMember.dob = dtpDOB.Text.ToString
        oneMember.email = txtEmail.Text
        oneMember.phone = txtPhone.Text
        oneMember.instrument = cmbInstrument.Text
        oneMember.level = cmbLevel.Text
        oneMember.photoPerm = chkPhotoPerm.Checked
        oneMember.groups = groups
        oneMember.contName = txtContName.Text
        oneMember.contPhone = txtContPhone.Text
        oneMember.role = cmbRole.Text

        // storing structure in file
        OPEN FILE(1, "players.dat", OpenMode.Random,,, Len(oneMember))
        FILE WRITE(1, oneMember, currentRecord)
        CLOSE FILE(1)
        OUTPUT "Player details updated"
        dgvRefresh()     // call subroutine that outputs file in dgv
End Sub
```

Search for data

This linear search algorithm will be used in many areas of the system either as a search for a record containing a certain field, in the instruments and music screens, or to display the event bookings on a date selected in the calendar by the user.

The pseudocode below searches for an instrument inputted by the user. Each record is read from the file and checked if it contains the search item. If the search item is found. The instrument is outputted in the DataGridView. It also checks if the instrument has a holderID stored or not. If it does, before the instrument is added to the DataGridView, the players file is opened and the holderName is found by searching for the holderID.

```
Sub search
    INPUT searchItem
    DECLARE oneInstrument is instruments    // pointer to structure
    DECLARE oneMember is memberInfo
    DECLARE quantity is Integer = 0
    DECLARE totalRecordsMember is Integer
    DECLARE totalRecordsInstrument is Integer

    IF searchItem = "" THEN
        OUTPUT “Select an instrument to search"
    END IF
    OPEN FILE(1, "instruments.dat", OpenMode.Random,,, Len(oneInstrument))

    totalRecordsInstrument = LOF(1) / Len(oneInstrument)

    IF totalRecordsInstrument = 0 THEN
        OUTPUT “No instruments stored"
    END IF

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

        IF oneInstrument.instrument.Contains(searchItem) THEN
            IF oneInstrument.holderID = "" Or oneInstrument.holderID = “” THEN
                dgvInstrumentSearch.Rows.Add(oneInstrument.instrumentID.Trim(),
oneInstrument.serialNumber.Trim(), oneInstrument.name.Trim(), oneInstrument.instrument.Trim(),
oneInstrument.holderID.Trim(), "", oneInstrument.serviceDate)
                quantity += 1
            ELSE    // if holderID stored, find holderName in players file
                FOR i = 1 To totalRecordsMember
```

```
                    FileGet(2, oneMember)
                    IF oneMember.id.Contains(oneInstrument.holderID) THEN
                        dgvInstrumentSearch.Rows.Add(oneInstrument.instr
                    umentID.Trim(), oneInstrument.serialNumber.Trim(),
                    oneInstrument.name.Trim(),
                    oneInstrument.instrument.Trim(),
                    oneInstrument.holderID.Trim(), oneMember.name.Trim(),
                    oneInstrument.serviceDate)
                        quantity += 1
                        Exit FOR
                    END IF
                NEXT
            END IF
        END IF
        CLOSE FILE(2)
    NEXT
    CLOSE FILE(1)

    txtQuantity.Text = quantity        // output number of instruments found
    CLOSE FILE(1)
    IF quantity = 0 THEN
        OUTPUT "No available instruments found"
    END IF
END Sub
```

Sort data

```
// bubble sort on dates so they are in descending order
DECLARE swapped is Boolean = TRUE
DECLARE temp is String
DECLARE n is Integer = recordCount - 1
DECLARE index is Integer

WHILE swapped = TRUE
    swapped = FALSE
    FOR index = 0 To n - 1
    // if a date is greater than the following date, swap dates, counts and totalRead at the index
        IF dates(index) > dates(index + 1) THEN
            temp = dates(index)
            dates(index) = dates(index + 1)
            dates(index + 1) = temp

            temp = counts(index)
            counts(index) = counts(index + 1)
            counts(index + 1) = temp

            temp = totalRead(index)
            totalRead(index) = totalRead(index + 1)
            totalRead(index + 1) = temp

            swapped = TRUE
        END IF
    NEXT
END WHILE
```

Change password

```
Sub changePassword
    DECLARE password1, password2 is string
    DECLARE userRecord is integer
    INPUT password1, password2
    IF password1 = "" THEN
        OUTPUT "New password must be entered"
        EXIT Sub

    ELSE IF password2 = "" THEN
        OUTPUT "New password must be entered twice"
        EXIT Sub

    ELSE IF password1.equals(password2) = FALSE THEN
        OUTPUT "Passwords entered do not match"
        EXIT Sub

    ELSE IF Len(txtPassword.text) < 8 THEN
        OUTPUT "Password must be at least 8 characters long"
        EXIT Sub

    ELSE IF Len(txtPassword.Text) > 20 THEN
        OUTPUT "Maximum password length is 20 characters"
        EXIT Sub
    END IF

    DECLARE oneMember is memberInfo    // pointer to structure
    OPEN FILE(1, "players.dat", OpenMode.Random,,, Len(oneMember))
    GET FILE(1, oneMember, frmLogin.userRecord)
    oneMember.password = password1

    DECLARE role as string
    IF oneMember.role.Contains("Conductor") THEN
        Role  = "conductor"

    ELSE IF oneMember.role.Contains("Librarian") THEN
        Role  = "librarian"

    ELSE IF oneMember.role.Contains("Instrument steward") THEN
        Role  = "instruments"

    ELSE IF oneMember.role.Contains("Event coordinator") THEN
        Role  = "events"
    ELSE IF oneMember.role.Contains("Committee member") THEN
        Role  = "committee member"

    ELSE IF oneMember.role.Contains("Treasurer") THEN
        Role  = "treasurer"

    ELSE IF oneMember.role.Contains("Player") THEN
        Role  = "player"

    ELSE IF oneMember.role.Contains("Dep") THEN
        Role  = "dep"
    END IF

    FILE WRITE(1, oneMember, frmLogin.userRecord)
    CLOSE FILE(1)

    OUTPUT "Password updated"
END Sub
```

Login to the system

```
Sub login
    DECLARE name, password, userID, role is string
    DECLARE userRecord is integer
    INPUT name, password
    IF name  = "" THEN
        OUTPUT "Enter your name”
        EXIT Sub
    ELSE IF password = "" THEN
        OUTPUT “Enter password”
        EXIT Sub
    END IF
    DECLARE oneMember is memberInfo     // pointer to structure
    DECLARE found is Boolean

    OPEN FILE(1, "players.dat", OpenMode.Random,,, Len(oneMember))

    DECLARE totalRecords is Integer = LOF(1) / Len(oneMember)
    DECLARE index is Integer
    FOR index = 1 To totalRecords
        GET FILE(1, oneMember)

        // find member in file
        IF oneMember.password.Contains(password) THEN
            IF oneMember.name.Contains(name) THEN
                found = True
                userID = oneMember.id
                userRecord = index

                IF Trim(oneMember.password) = Trim(oneMember.id) THEN
                    frmChangePassword.Show()
                    Me.CLOSE()
                    CLOSE FILE(1)
                    EXIT Sub
                END IF

                // find member access
                IF oneMember.role.Contains("Conductor") THEN
                    role = "conductor"

                ELSE IF oneMember.role.Contains("Librarian") THEN
                    role = "librarian"

                ELSE IF oneMember.role.Contains("Instrument steward") THEN
```

```
                        role = "instruments"

                    ELSE IF oneMember.role.Contains("Event coordinator") THEN
                        role = "events"

                    ELSE IF oneMember.role.Contains("Committee member") THEN
                        role = "committee member"

                    ELSE IF oneMember.role.Contains("Treasurer") THEN
                        role = "treasurer"

                    ELSE IF oneMember.role.Contains("Player") THEN
                        role = "player"

                    ELSE IF oneMember.role.Contains("Dep") THEN
                        role = "dep"
                    END IF
                END IF
            END IF
    NEXT

    CLOSE FILE(1)
    IF found = FALSE THEN
        OUTPUT "Incorrect login details."
    ELSE
        frmHome.Show()
    END IF
END sub
```

Reset password

```
Sub ResetPassword
    DECLARE name, password is string
    INPUT name, password
    IF name  = "" THEN
        OUTPUT Name must be entered in order to reset password"
        EXIT Sub
    END IF
    oneMember is memberInfo        // pointer to structure
    DECLARE found is boolean =  FALSE
    FileOpen(1, "players.dat", OpenMode.Random,,, Len(oneMember))
    DECLARE totalRecords is Integer = LOF(1) / Len(oneMember)
    FOR index = 1 To totalRecords
        GET FILE(1, oneMember)
        IF Trim(oneMember.name) = name THEN
            oneMember.password = oneMember.id   // save password
            FILE WRITE(1, oneMember, index) // 'update file
            CLOSE FILE(1)
            OUTPUT "Password reset to ID"
            found = True
            EXIT FOR
        END IF
    NEXT
    FileClose(1)
    IF found = FALSE THEN
        OUTPUT "Name not found in file"
    END IF
END Sub
```

Managing checkbox data

The following algorithms are used to manage data that is inputted and outputted using checkboxes. Checkboxes are used to record the groups a player is a member of as one player can be a member of many groups. The following algorithms are used across the system in areas that require groups to be inputted.

This algorithm converts the boolean inputs from the checkboxes into one string so that the groups can be viewed in the DataGridView in an easier format for the user. If the parameter is true, a string is stored in a temporary file. Once the value in each parameter has been processed, the file is opened, and each item in the file is read into a string separated by commas. This is returned to the subroutine that called the function.

```
Function groupsToString(psb is boolean, pytb is boolean, pbb is boolean, starters is boolean)
    INPUT psb, pytb, pbb, starters
    // convert the boolean input into a string so the groups can be displayed in a table
    DECLARE totalGroups is String
        IF psb = FALSE THEN
            IF pytb = FALSE THEN
                IF pbb = FALSE THEN
                    IF starters = FALSE THEN
                        OUTPUT "At least one group must be selected"
                        totalGroups = "FALSE"
                        RETURN totalGroups
                    END IF
                END IF
            END IF
        END IF

        OPEN FILE "groupsTemp.csv"
        IF psb = TRUE THEN
            FILE WRITE ("PSB", TRUE)
        END IF
        IF pytb = TRUE THEN
            FILE WRITE ("PYTB", TRUE)
        END IF

        IF pbb = TRUE THEN
            FILE WRITE ("PBB", TRUE)
        END IF

        IF starters = TRUE THEN
            FILE WRITE ("Starters", TRUE)
        END IF
        CLOSE FILE "groupsTemp.csv"
        OPEN FILE "groupsTemp.csv"
        DO UNTIL EOF
            IF totalGroups = "" THEN
                totalGroups = reader.ReadLine
            ELSE
                totalGroups = totalGroups & ", " & reader.ReadLine
            END IF
        LOOP
        CLOSE FILE "groupsTemp.csv"
        DELETE FILE("groupsTemp.csv")
        RETURN totalGroups
    END Function
```

These functions convert groups that are stored as a string to boolean. Each function takes the string read from the file and if the string contains the group, true is returned. This can then be used by the subroutine to output the data in the checkboxes.

```
// converts stored groups string into boolean so can be shown in checkboxes
Function tickPSB(data)
    DECLARE found is Boolean = FALSE
    IF data.contains("PSB") THEN
        found = TRUE
    END IF
    RETURN found
END Function
Function tickPYTB(data)
    DECLARE array is Array = data.split
    DECLARE found is Boolean = FALSE
    FOR i = 0 To 6
        IF array(i).contains("PYTB") THEN
            found = TRUE
        END IF
    NEXT
    RETURN found
END Function
Function tickPBB(data)
    DECLARE array is Array = data.split
    DECLARE found is Boolean = FALSE
    FOR i = 0 To 6
        IF array(i).contains("PBB") THEN
            found = TRUE
        END IF
    NEXT
    RETURN found
END Function
Function tickStarters(data)
    DECLARE array is Array = data.split
    DECLARE found is Boolean = FALSE
        FOR i = 0 To 6
            IF array(i).contains("Starters") THEN
                found = TRUE
            END IF
        NEXT
    RETURN found
END Function
```