# Design

## Input & Output - Specify, design and document screen layouts, reports and other forms of input and output required to create the user interface.

## Design and document all data structures that will be required to produce the output for the solution to the problem together with the method of accessing the data in that data structure.

## Ensure that all data entered into the system is valid.

## Design programming routines to be used to handle and process data within the proposed solution to achieve each objective.

## Document these designs using a structured convention such as pseudo-code.

Design Overview

The system I am going to develop will have a suitable and justified User Interface. The system I am going to build will have a graphical user interface. I have chosen to develop this UI because theoretically it should be easier to implement the solution than on a CLI due to the complexity of the program I am designing and its needs. A GUI is also a lot easier to maintain, if new features were to be added after the initial systems development done my myself, it would be much easier for the user to be able to find and use these new features in a GUI as it would not compromise the existing features. A GUI will allow me to develop an intuitive but yet consistent system. I will design my program with the appearance in mind, I will follow the standard layout of my application, with menu bars along the top and down the left-hand side. This layout makes the navigation and usability have a sense of familiarity so that the users feel more confidant and are able to repeat their actions. I will make the features used most frequently the most prominent throughout, making the user feel more assured and courageous with the system.

A few libraries I will use after some research include the following:

* Tkinter – Tkinter will be the library I will use to create my GUI along with any standard GUI features including Labels, Entry Boxes or Drop lists.
* Report Lab – I will use this to Generate a transport enquiry which will be populated with data the Pipe Major has entered into the Book Transport Form. This will then be sent to the Bus Company where they will do all further communication with the Pipe Major Directly.
* Email – I will use this library to send an email from a RSPABNI email account to the Bus Company. I will create a generic body of text with a subject to include the name of the band that would like to book transport. I will also have to be able to attach the generated PDF from report Lab to the email as well.
* SQLite3 – This library will allow me store information in a database file. I am able to retrieve, amend and save data to the database file using SQL commands with this library

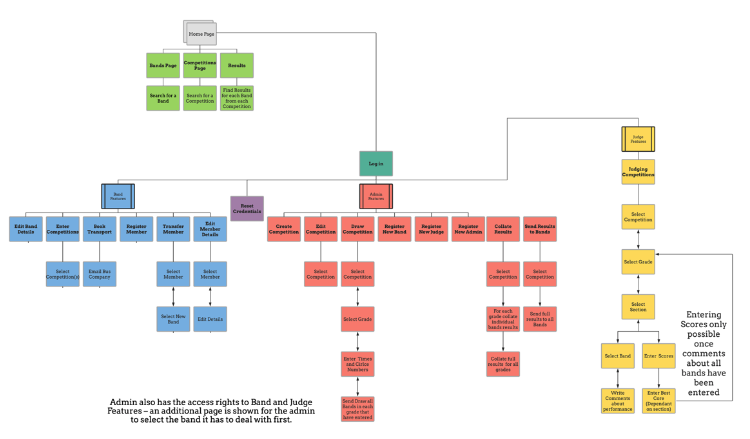
My desk-based research in my investigation has provided me with good solutions to problems I am going to face during development, how I end up developing my code will be reflected in the research I have done. My research has also provided me with the necessary fields which I need to ask the user for to complete certain tasks – I will explain these for each objective when I get to them.

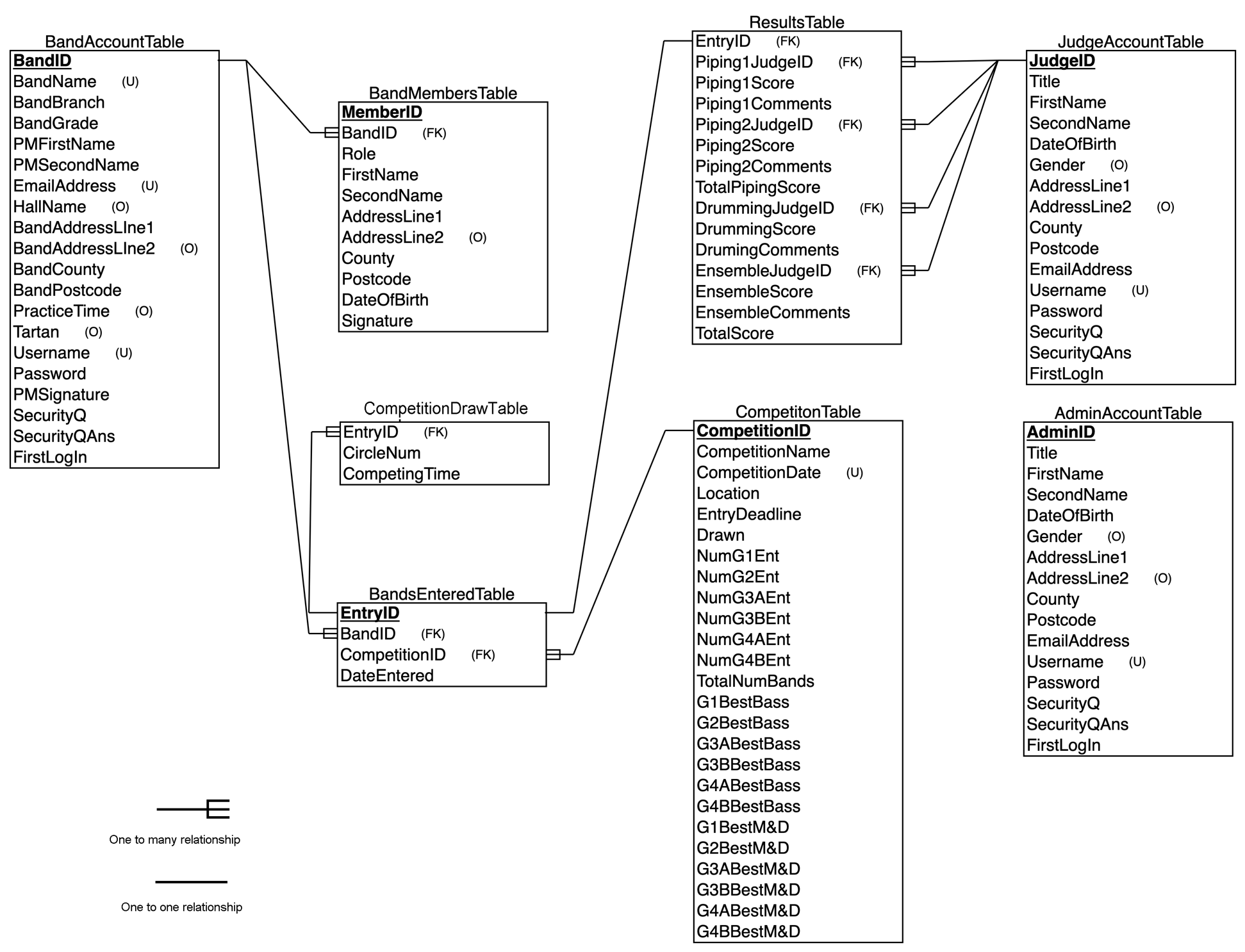
I have included a structured DFD & ERD diagram below to show the scope of my project and breakdown of my system of the data that I will need to store. I will describe how I intend to manipulate my data structure (database) as and when new data is going to be added in the objective where it is dealt with and make reference to my data structures throughout.

I have split my project down into my objectives. For each objective I will give a quick overview of what it involves, also within each objective I will fully:

* Describe data to be inputted and outputted from the system.
* Consider the reasons for the data to be included in my database and outputs.
* Describe all files, data structures and methods of access.
* Describe validation routines to be carried out on raw data.
* Describe the processing routines.

*\*Please find a higher quality version of DFD as Appendix\_i.png*





I have created a data dictionary for each of my tables within my database. Along with the standard requirements I will make reference to why I have included them giving their purpose and how I will access the data.

Key –

Green box means yes

Red box means no

*Figure 3.0.1 – Admin Details.*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table Name:** AdminAccountTable | | **Estimated number of records:** | | 10 - 15 | | **Foreign Keys:** | |  |
| **Table Purpose:** Contains basic details about administrators of the system | | **Type of data structure and how will it be accessed** | | Database storing data as records. Accessed using SQLite3, Insert Set and Select where Commands | | **Composite Keys** | |  |
| **Field Name:** | **Primary Key?** | **Field Type:** | **Field Length in bytes:** | **Required at registration?** | **Default Value:** | **Validation methods on entry:** | **Example Data** | **Field Purpose** |
|  |  |  |  |  |  |  |  |  |
| **AdminID** |  | Auto Number |  |  | N/A | * N/A as auto increments the primary key | 12 | Used to uniquely identify Admin accounts. |
| **Title** |  | String | 5 |  | N/A | * Dropdown list | Mrs | Must be stored as part of the RSPBANI policy as point of contact details. |
| **FirstName** |  | String | 20 |  | N/A | * Presence check * Alphabetic check | Adam | Must be stored as part of the RSPBANI policy as point of contact details. |
| **SecondName** |  | String | 20 |  | N/A | * Presence check * Alphabetic check | Faucett | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| **AddressLineOne** |  | String | 60 |  | N/A | * Presence check | Belfast Road | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| **AddressLineTwo** |  | String | 60 |  | “” | * Alphabetical check | Aldergrove | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| **County** |  | String | 20 |  | N/A | * Dropdown list | Down | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| **Postcode** |  | String | 7 |  | N/A | * Presence * Format check – LLNN NLL or LLNNLL * Length check – 7/8 | BT38 8HJ | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| **DOB** |  | String | 10 |  | N/A | * Date format – DD/MM/YYYY | 12/09/1960 | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| **Gender** |  | String | 6 |  | N/A | * Radio Button (Male, Female, other) | Male | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| **Username** | **Unique** | String | 20 |  | N/A | * Presence check * Uniqueness check | adamF687 | Username is required to identify the user when logging on to the system. |
| **Password** |  | String | 20 |  | N/A | * Presence check * Length check - >8 * Character check >=1 [A-Z] & >= [a-z] * Special character check | @dAmF10 | To also identify the user but also make the account secure. |
| **EmailAddress** |  | String | 60 |  | N/A | * Presence check * Format check – string@string.string | Adam.f687@gmail .com | The email address is required s it is asked to be confirmed when the user wants to change their password. |
| **SecurityQ** |  | String | 40 |  | “” | * Presence Check * Dropdown list | What is your favourite team? | The Security Question and answer is used as proof of the person so as they can identify themselves before, they change their password as a layer of protection. |
| **SecurityAns** |  | String | 40 |  | “” | * Presence Check | Liverpool FC |
| **FirstLogin** |  | Boolean | 1 |  | False | * True or False | False | This is used and checked when this user logs in. If they haven’t logged in before it will allow the program to ask them to change their password. |

*Figure 3.0.2 – Band Details.*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table Name:** BandAccountTable | | **Type of data structure and how it will be accessed** | | Database storing data as records. Accessed using SQLite3, Insert Set and Select where Commands | | | **Foreign Keys:** |  |
| **Table Purpose:** Contains all the required information about the band to allow it to enter competitions and display about it. | | | | **Estimated number of records** | | 100 - 200 | **Composite Keys:** |  |
| **Field Name:** | **Primary Key?** | **Data Type:** | **Field Length in bytes:** | **Required?** | **Default Value:** | **Validation methods used:** | **Example Data:** | **Field purpose:** |
|  |  |  |  |  |  |  |  |  |
| BandID |  | Auto Number |  |  | N/A | * Presence check | 25 | Used for all processing stages involved with bands as this uniquely identified every band. Eg. Registering Members, Entering competitions and editing band details |
| Grade |  | String | 5 |  | N/A | * Presence check * Dropdown list | 3B | This is the grade of the Band used in many processing stages |
| BandName |  | String | 30 |  | N/A | * Presence check * Alphabetical check | Ballydonaghy | This is the BandName which is used for output instead of processing |
| BranchSection |  | String | 20 |  | N/A | * Presence * Dropdown list | Co.Antrim | Information about the band that is displayed on the bands page so the public can find out about the band. |
| PMFirstName |  | String | 20 |  | N/A | * Presence check * Alphabetic check | Andy | Information about the band that is displayed on the bands page so the public can find out about the band. |
| PMSecondName |  | String | 20 |  | N/A | * Presence check * Alphabetic check | Baker | Information about the band that is displayed on the bands page so the public can find out about the band. |
| HallName |  | String | 20 |  | “” | * Alphabetical check | The Mount | Information about the band that is displayed on the bands page so the public can find out about the band. |
| AddressLineOne |  | String | 60 |  | N/A | * Presence check | Garlandstwown Road | Information about the band that is displayed on the bands page so the public can find out about the band. |
| AddressLineTwo |  | String | 60 |  | “” | * Alphabetical check | Crumlin | Information about the band that is displayed on the bands page so the public can find out about the band. |
| County |  | String | 30 |  | N/A | * Presence check | Antrim | Information about the band that is displayed on the bands page so the public can find out about the band. |
| Postcode |  | String | 8 |  | N/A | * Presence Check * Format check– LLNN NLL or LLNNLL * Length check – 7/8 | BT29 7HJ | Information about the band that is displayed on the bands page so the public can find out about the band. |
| PracticeTime |  | String | 30 |  | “Unknown” | * None | 7:30pm – 10:00pm | Information about the band that is displayed on the bands page so the public can find out about the band. |
| Tartan |  | String | 30 |  | “Unknown” | * Alphabetical check | Royal Stewart | Information about the band that is displayed on the bands page so the public can find out about the band. |
| EmailAddress | **Unique** | string | 60 |  | N/A | * Presence check * Format check – string@string.string | AndyB987@gmail.com | This is the Pipe Majors email. It is used to send confirmations of entering competitions and also as a field to the bus company so that they can arrange the rest independently of RSPBANI & change password |
| Username | **Unique** | String | 20 |  | N/A | * Presence check * Uniqueness check | ABaker987 | Username is required to identify the user when logging on to the system. |
| Password |  | String | 20 |  | N/A | * Presence check * Length check - >8 * Character check >=1 [A-Z] & >= [a-z] * Special character check | Pictur@THIS78 | To also identify the user but also make the account secure. |
| **SecurityQ** |  | String | 40 |  | “” | * Presence Check * Dropdown list | In what town was your first job? | The Security Question and answer is used as proof of the person so as they can identify themselves before they change their password as a layer of protection. |
| **SecurityAns** |  | String | 40 |  | “” | * Presence Check | London |
| **FirstLogin** |  | Boolean | 1 |  | False | * True or False | False | This is used and checked when this user logs in. If they haven’t logged in before it will allow the program to ask them to change their password. |

*Figure 3.0.3 – Band Members’ Details*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table Name:** BandMembersTable | | **Type of data structure and how it will be accessed** | | Database storing data as records. Accessed using SQLite3, Insert Set and Select where Commands | | | **Foreign Keys:** | **BandID** |
| **Table Purpose:** Contains all the required information about thto allow it to enter competitions and display about it. | | | | **Estimated number of records** | | **~** 3000 | **Composite Keys:** |  |
| **Field Name:** | **Primary Key?** | **Data Type:** | **Field Length in bytes:** | **Required?** | **Default Value:** | **Validation methods used:** | **Example Data:** | **Field purpose:** |
|  |  |  |  |  |  |  |  |  |
| MemberID |  | Auto Number |  |  | N/A | * Presence check | 25 | Used for all processing stages involved with bands as this uniquely identified every band. Eg. Registering Members, Entering competitions and editing band details |
| BandID |  | Integer |  |  | N/A | * Presence Check | 2 | This is to link the member to a band. |
| Role |  | String | 20 |  | N/A | * Presence check * Dropdown list | Pipe Sergeant | This is the instrument that the band member plays within the band. |
| FirstName |  | String | 20 |  | N/A | * Presence check * Alphabetic check | Andy | Information about the band that is displayed on the bands page so the public can find out about the band. |
| SecondName |  | String | 20 |  | N/A | * Presence check * Alphabetic check | Baker | Information about the band that is displayed on the bands page so the public can find out about the band. |
| AddressLineOne |  | String | 60 |  | N/A | * Presence check | Garlandstwown Road | Information about the band that is displayed on the bands page so the public can find out about the band. |
| AddressLineTwo |  | String | 60 |  | “” | * Alphabetical check | Crumlin | Information about the band that is displayed on the bands page so the public can find out about the band. |
| County |  | String | 30 |  | N/A | * Presence check | Antrim | Information about the band that is displayed on the bands page so the public can find out about the band. |
| Postcode |  | String | 8 |  | N/A | * Presence Check * Format check– LLNN NLL or LLNNLL * Length check – 7/8 | BT29 7HJ | Information about the band that is displayed on the bands page so the public can find out about the band. |
| **DOB** |  | String | 10 |  | N/A | * Date format – DD/MM/YYYY | 12/02/1989 | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| **Gender** |  | String | 6 |  | N/A | * Radio Button (Male, Female, other) | Male | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| Email Address | **Unique** | string | 60 |  | N/A | * Presence check * Format check – string@string. | AndyB987@gmail.com | Their email address is required as the method of contact. |
| Signature |  | string | 40 |  | N/A | * Presence check | Andy Baker | Their signature is required so confirm that they agree for the information given is correct and the RSPBANI have permission to store and use it. |

Figure 3.0.4 – Judges’ Details.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table Name:** JudgeAccountTable | | **Estimated number of records:** | | 10 - 15 | | **Foreign Keys:** | |  |
| **Table Purpose:** Contains basic details about the Judges of the association | | **Type of data structure and how will it be accessed** | | Database storing data as records. Accessed using SQLite3, Insert Set and Select where Commands | | **Composite Keys** | |  |
| **Field Name:** | **Primary Key?** | **Field Type:** | **Field Length in bytes:** | **Required at registration?** | **Default Value:** | **Validation methods on entry:** | **Example Data** | **Field Purpose** |
|  |  |  |  |  |  |  |  |  |
| **JudgeID** |  | Auto Number |  |  | N/A | N/A as auto increments the primary key | 12 | Used to uniquely identify Judge accounts. |
| **Title** |  | String | 5 |  | N/A | * Dropdown list | Mrs | Must be stored as part of the RSPBANI policy as point of contact details. |
| **FirstName** |  | String | 20 |  | N/A | * Presence check * Alphabetic check | Adam | Must be stored as part of the RSPBANI policy as point of contact details. |
| **SecondName** |  | String | 20 |  | N/A | * Presence check * Alphabetic check | Faucett | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| **AddressLineOne** |  | String | 60 |  | N/A | * Presence check | Belfast Road | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| **AddressLineTwo** |  | String | 60 |  | “” | * Alphabetical check | Aldergrove | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| **County** |  | String | 30 |  | N/A | * Dropdown list | Down | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| **Postcode** |  | String | 7 |  | N/A | * Presence * Format check – LLNN NLL or LLNNLL * Length check – 7/8 | BT38 8HJ | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| **DOB** |  | String | 10 |  | N/A | * Date format – DD/MM/YYYY | 12/09/1960 | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| **Gender** |  | String | 6 |  | N/A | * Radio Button (Male, Female, other) | Male | Must be stored as part of the RSPBANI policy as point of contact and to identify the person linked to the account. |
| **Username** | **Unique** | String | 20 |  | N/A | * Presence check * Uniqueness check | adamF687 | Username is required to identify the user when logging on to the system. |
| **Password** |  | String | 20 |  | N/A | * Presence check * Length check - >8 * Character check >=1 [A-Z] & >= [a-z] * Special character check | @dAmF10 | To also identify the user but also make the account secure. |
| **EmailAddress** |  | String | 60 |  | N/A | * Presence check * Format check – string@string.string | Adam.f687@gmail .com | The email address is required when user is asked to be confirmed when the user wants to change their password. |
| **SecurityQ** |  | String | 40 |  | “” | * Presence Check * Dropdown list | What is your favourite team? | The Security Question and answer is used as proof of the person so as they can identify themselves before, they change their password as a layer of protection. |
| **SecurityAns** |  | String | 40 |  | “” | * Presence Check | Liverpool FC |
| **FirstLogin** |  | Boolean | 1 |  | False | * True or False | False | This is used and checked when this user logs in. If they haven’t logged in before it will allow the program to ask them to change their password. |

Figure 3.0.5 – Competitions

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table Name:** CompetitionTable | | **Estimated number of records:** | | **~** 200 | | **Foreign Keys:** | |  |
| **Table Purpose:** Contains basic details about the Judges of the association | | **Type of data structure and how will it be accessed** | | Database storing data as records. Accessed using SQLite3, Insert Set and Select where Commands | | **Composite Keys** | |  |
| **Field Name:** | **Primary Key?** | **Field Type:** | **Field Length in bytes:** | **Required at creation of competition?** | **Default Value:** | **Validation methods on entry:** | **Example Data** | **Field Purpose** |
|  |  |  |  |  |  |  |  |  |
| **CompetitionID** |  | Auto Number |  |  | N/A | N/A as auto increments the primary key | 12 | Used to uniquely identify each competition held in different years. |
| **CompetitionName** |  | String | 50 |  | N/A | * Dropdown list | “Co.Antrim Championships” | Displayed to the user when they are selecting a competition as they would not be able to distinguish competitions from their ID |
| **CompetitionDate** |  | Date | 5 |  | N/A | * Presence check * Alphabetic check | 15/08/19 | This is also displayed to the user so that they know they date of the competition |
| **Location** |  | String | 50 |  | N/A | * Presence check * Alphabetic check | “Antrim Castle Gardens” | Displayed to the user so that they know where the competition is being held |
| **EntryDeadline** |  | Date | 5 |  | N/A | * Presence check | 15/06/19 | This is used by the program to only allow the user to enter competitions before this date. After this date the competition will appear on the admin side to be able to draw it. It is also displayed to bans on the competition page so they know when they have to enter each competition by. |
| **Drawn** |  | Boolean | 1 |  | False | * Alphabetical check | True | This variable is incremented to true after the deadline and once the admin has drawn the competition. |
| **NumG1Ent** |  | Integer |  |  | 0 | * N/A – calculated by program | 1 | This stores the number of bands in Grade 1 for this competition. This is only calculated once the draw has taken place. It is used when the details about the competitions has been updated to include the draw. |
| **NumG2Ent** |  | Integer |  |  | 0 | N/A – calculated by program | 6 | This stores the number of bands in Grade 2 for this competition. This is only calculated once the draw has taken place. It is used when the details about the competitions has been updated to include the draw. |
| **NumG3AEnt** |  | Integer |  |  | 0 | N/A – calculated by program | 10 | This stores the number of bands in Grade 3A for this competition. This is only calculated once the draw has taken place. It is used when the details about the competitions has been updated to include the draw. |
| **NumG3BEnt** |  | Integer |  |  | 0 | * N/A – calculated by program | 5 | This stores the number of bands in Grade 3B for this competition. This is only calculated once the draw has taken place. It is used when the details about the competitions has been updated to include the draw. |
| **NumG4AEnt** | **Unique** | Integer |  |  | 0 | * N/A – calculated by program | 6 | This stores the number of bands in Grade 4A for this competition. This is only calculated once the draw has taken place. It is used when the details about the competitions has been updated to include the draw. |
| **NumG4BEnt** |  | Integer |  |  | 0 | N/A – calculated by program | 20 | This stores the number of bands in Grade 4B for this competition. This is only calculated once the draw has taken place. It is used when the details about the competitions has been updated to include the draw. |
| **TotalNumBands** |  | String | 3 |  | 0 | * N/A – calculated by program | 66 | This adds up the number of bands entered into each grade to be displayed on the Competitions page. |
| **G1BestBass** |  | String | 40 |  | “” | * Dropdown list of bands entered in the specific grade. | Field Marshal Montgomery | The value of this is set to “” when the competition is created. Once the drumming judge has entered the comments for each band in the specific grade. They then must select the band with the best bass section, and this is updated to reflect that. It is then used to be displayed on the Results page |
| **G2BestBass** |  | String | 40 |  | “” | * Dropdown list of bands entered in the specific grade. | PSNI |
| **G3ABestBass** |  | String | 40 |  | “” | * Dropdown list of bands entered in the specific grade. | Broughshane |
| **G3BBestBass** |  | String | 40 |  | “” | * Dropdown list of bands entered in the specific grade. | McNeilstown |
| **G4ABestBass** |  | String | 40 |  | “” | * Dropdown list of bands entered in the specific grade. | Gransha |
| **G4BBestBass** |  | String | 40 |  | “” | * Dropdown list of bands entered in the specific grade. | Broughshane |
| **G1BestM&D** |  | String | 40 |  | “” | * Dropdown list of bands entered in the specific grade. | Field Marshal Montgomery |
| **G2BestM&D** |  | String | 40 |  | “” | * Dropdown list of bands entered in the specific grade. | PSNI |
| **G3ABestM&D** |  | String | 40 |  | “” | * Dropdown list of bands entered in the specific grade. | Broughshane | The value of this is set to “” when the competition is created. Once the ensemble judge has entered the comments for each band in the specific grade. They then must select the band with the best march and deportment, and this is updated to reflect that. It is then used to be displayed on the Results page |
| **G3BBestM&D** |  | String | 40 |  | “” | * Dropdown list of bands entered in the specific grade. | McNeilstown |
| **G4ABestM&D** |  | String | 40 |  | “” | * Dropdown list of bands entered in the specific grade. | Gransha |
| **G4BBestM&D** |  | String | 40 |  | “” | * Dropdown list of bands entered in the specific grade. | Broughshane |

*Figure 3.0.6 – Bands that have entered competitions*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table Name:** BandsEnteredTable | | **Estimated number of records:** | | **~** 1000 | | **Foreign Keys:** | |  |
| **Table Purpose:** Contains a record for competition that each band enters | | **Type of data structure and how will it be accessed** | | Database storing data as records. Accessed using SQLite3, Insert Set and Select where Commands | | **Composite Keys** | | **BandID, CompetitionID** |
| **Field Name:** | **Primary Key?** | **Field Type:** | **Field Length in bytes:** | **Default Value:** | **Validation methods on entry:** | | **Example Data** | **Field Purpose:** |
|  |  |  |  |  |  | |  |  |
| **EntryID** |  | Auto Number |  | N/A | N/A as auto increments the primary key | | 100 | Used to uniquely identify each entry. |
| **BandID** |  | Integer |  | N/A | * N/A as this field is just queried from the Band Account Table of the either the Band Logged in or the band the admin has selected | | 15 | The primary key, BandID, is included along with the primary key, competitonID. This creates a composite key to ensure that each record is unique and can be easily found when queried |
| **CompetitionID** |  | Integer |  | N/A | * N/A as this field is just queried from the CompetitionsTable of the either the Band Logged in or the band the admin has selected | | 8 |
| **DateEntered** |  | Date | 20 | N/A | * Format check – DD/MM/YYYY | | 12/05/18 | This field will be used to store the date of entry to be outputted in the confirmation email that will be sent to the PM when he enters a competition. |

*Figure 3.0.7 – Bands that have entered competitions*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table Name:** CompetitionDrawTable | | **Estimated number of records:** | | **~** 1000 | | **Foreign Keys:** | |  |
| **Table Purpose:** Contains a record for competition that each band enters | | **Type of data structure and how will it be accessed** | | Database storing data as records. Accessed using SQLite3, Insert Set and Select where Commands | | **Composite Keys:** | | **EntryID; BandID & CompetitionID** |
| **Field Name:** | **Primary Key?** | **Field Type:** | **Field Length in bytes:** | **Default Value:** | **Validation methods on entry:** | | **Example Data** | **Field Purpose:** |
|  |  |  |  |  |  | |  |  |
| **EntryID** |  | Integer | 4 | N/A | * N/A as this field is just queried from the Bands EnteredTable of the either the Band Logged in or the band the admin has selected | | 8 | This composite key is this entities primary key which uniquely identifies the band at a competition |
| **CircleNum** |  | Integer |  | N/A | * Range check: 1-10 | | 5 | This is the Circle number that will be outputted to the user once the drawn has taken place |
| **CompetingTime** |  | String | 5 | N/A | * Format check – NN:NN | | 11:15 | This is the Circle number that will be outputted to the user once the drawn has taken place |

Figure 3.0.8 – Results

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table Name:** ResultsTable | | **Estimated number of records:** | | **~** 1000 | | **Foreign Keys:** | |  |
| **Table Purpose:** Records each bands results from every judge per competition and is used to calculate and store their final score. | | **Type of data structure and how will it be accessed** | | Database storing data as records. Accessed using SQLite3, Insert Set and Select where Commands | | **Composite Keys:** | | **EntryID; BandID & CompetitionID** |
| **Field Name:** | **Primary Key?** | **Field Type:** | **Field Length in bytes:** | **Default Value:** | **Validation methods on entry:** | | **Example Data** | **Field Purpose** |
|  |  |  |  |  |  | |  |  |
| **EntryID** |  | Auto Number |  | N/A | N/A as this field is just queried from the Bands Entered Table of the either the Band Logged in or the band the admin has selected | | 100 | To identify a bands result at a specific competition |
| **Piping1JudgeID** |  | Integer |  | “” | * N/A as this field is just queried from the Judge Account Table of the either the Judge Logged in or the judge the admin has selected | | 5 | The ID of the Judge who graded this section of the band at the competition. |
| **Piping1Score** |  | Integer |  | “” | * Range check: 1- Number of bands entered the grade | | 1 | The store the score the judge gives. |
| **Piping1Comments** |  | String | ~ | “” | * Presence Check | | “Their comments” | To store the comments the judge gives them for their performance. |
| **Piping2JudgeID** |  | Integer |  | “” | * N/A as this field is just queried from the Judge Account Table of the either the Judge Logged in or the judge the admin has selected | | 5 | The ID of the Judge who graded this section of the band at the competition. |
| **Piping2Score** |  | Integer |  | “” | * Range check: 1- number of bands entered the grade | | 1 | The store the score the judge gives. |
| **Piping2Comments** |  | String | ~ | “” | * Presence Check | | “Their comments” | To store the comments the judge gives them for their performance. |
| **TotalPipingScore** |  | Integer |  | “” | * Range: 2 – 2x Numbers of bands Entered the Grade | | 2 | This used to store the total of the piping scores |
| **DrumingJudgeID** |  | Integer |  | “” | * N/A as this field is just queried from the Judge Account Table of the either the Judge Logged in or the judge the admin has selected | | 5 | The ID of the Judge who graded this section of the band at the competition. |
| **DrummingScore** |  | Integer |  | “” | * Range check: 1- number of bands entered the grade | | 1 | The store the score the judge gives. |
| **DrummingComments** |  | String | ~ | “” | * Presence Check | | “Their comments” | To store the comments the judge gives them for their performance. |
| **EnsembleJudgeID** |  | Integer |  | “” | * N/A as this field is just queried from the Judge Account Table of the either the Judge Logged in or the judge the admin has selected | | 5 | The ID of the Judge who graded this section of the band at the competition. |
| **EnsembleScore** |  | Integer |  | “” | * Range: 1 – Number Bands Entered the grade | | 1 | The store the score the judge gives. |
| **EnsembleComments** |  | String | ~ | “” | * Presence Check | | “Their comments” | To store the comments the judge gives them for their performance. |
| **TotalScore** |  | Integer |  | “” | * Range: 4 – 4x Number of bands entered the grade. | | 5 | This used to store the total score of all the judges. |

As a whole my system has been designed with the colours and scheme of the association in mind. I have chosen Avro as the font that I am going to be implementing through my solution. I could not show this font in my interface designs due to limitations of the software I used. Most of the text though my system will be black on the white background. This keeps my system aesthetically pleasing and makes it easy to navigate.

Avro is the associations main font which they use in their current outputs and any documentation they produce. The size of the font will be determined by the importance of the text. The font will be size 32 for each page heading. The rest of the font will be of a smaller size and will depend on how much space there is available, however as a rule of thumb I will try to aim for size 10-12.

Throughout my program I have used coloured buttons to highlight different functionality. I have used Green buttons to distinguish these as main navigational buttons which will bring you to another page. Buttons which are blue in colour are involved in a feature in my system. These blue buttons will allow a user to work their way through a feature and save changes to data being stored on the system.

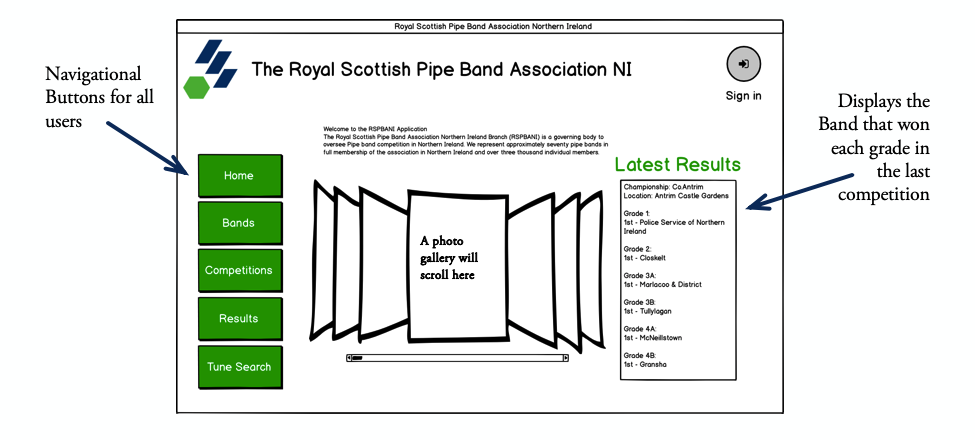
The hex code for the association’s colours are as followed:

Green - #009e0f

Blue - #073763

I will design my interface according to the objectives stated in the investigation phase, for each I will give a quick introduction into the screen, annotate the interface design and outline any processing stages that may exist to show you how data in my system will be handled and processed.

**“All users of the system on launch will be brought to the home page.”**



When the user runs the application, this home page will be presented to them. This Screen has been designed for easy and intuitive navigation. I chose big coloured buttons instead of icons because for the majority of the time, one of the biggest users of these pages will be bands members and spectators who will not be using the system very often and will not be trained in any way to be able use the system. The design features the photo gallery taking up most of this page.

The Latest Results section is very simply laid out to ensure there is no confusion about results but also acts as a quick go to feature to view the winners of each grade after a competition. Due to space and aesthetic feel of the Home Page, I did not want too much unnecessary band results which can be viewed under the results section. The information to be displayed in this section is queried from the database. The processing of how I will go about this is highlighted below.

The Green Button dashboard will be available throughout all pages within my system, to allow for easy navigation between them.

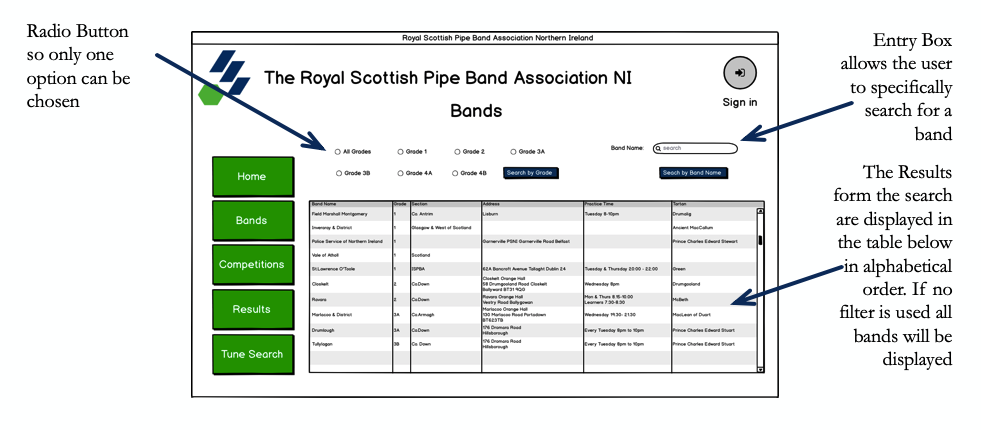
|  |  |
| --- | --- |
| Pseudocode for getting the latest competition results | DECLARE CompetitionID as Int  DECLARE CompetitionName as StringVar  DECLARE CompetitionLocation as StringVar  DECLARE G1Winner as StringVar  DECLARE G2Winner as StringVar  DECLARE G3AWinner as StringVar  DECLARE G3BWinner as StringVar  DECLARE G4AWinner as StringVar  DECLARE G4BWinner as StringVar  LatestCompetition = SELECT CompetitionID, CompetitionName, CompetitionLocation FROM Competitions WHERE Latest = “True”  IF LatestCompetition  CompetitionID = LatestCompetition[0]  CompetitionName = LatestCompetition [1]  CompetitionLocation = LatestCompetition[2]  ELSE  CompetitionID = “No latest Competition”  CompetitionName = “No latest Competition”  CompetitionLocation = “No latest Competition”  G1Winner = SELECT BandName FROM Results WHERE CompetitionID = CompetitionID AND Position = “1” AND Grade = “1”  G2Winner = SELECT BandName FROM Results WHERE CompetitionID = CompetitionID AND Position = “1” AND Grade = “2”  G3AWinner = SELECT BandName FROM Results WHERE CompetitionID = CompetitionID AND Position = “1” AND Grade = “3A”  G3BWinner = SELECT BandName FROM Results WHERE CompetitionID = CompetitionID AND Position = “1” AND Grade = “3B”  G4AWinner = SELECT BandName FROM Results WHERE CompetitionID = CompetitionID AND Position = “1” AND Grade = “4A”  G4BWinner = SELECT BandName FROM Results WHERE CompetitionID = CompetitionID AND Position = “1” AND Grade = “4B”  Populate textbox on GUI with  “CompetitionName” + CompetitionName  “CompetitionLocation” + CompetitionLocation  “Grade 1 – 1st place: ” + G1Winners  “Grade 2 – 1st place: ” + G2Winners  “Grade 3A – 1st place: ” + G3AWinners  “Grade 3B – 1st place: ” + G3BWinners  “Grade 4A – 1st place: ” + G4AWinners  “Grade 4B – 1st place: ” + G4BWinners  END |

**“All users have the ability to view and search for a band on the bands page.”**

Another page that guest users can view without necessary logging in is this Bands page. All users are able to search by multiple filters. The user will be able to view all the Bands in a certain grade by selecting the radio button they wish to search by and then pressing the related search button.

The selections are spaced out symmetrically which make it easier to find the filter you are looking for. There is a button to search by the selection you have entered. It will be in the blue of the association which will stand out so that the user will realise they must press it to filter to their results.

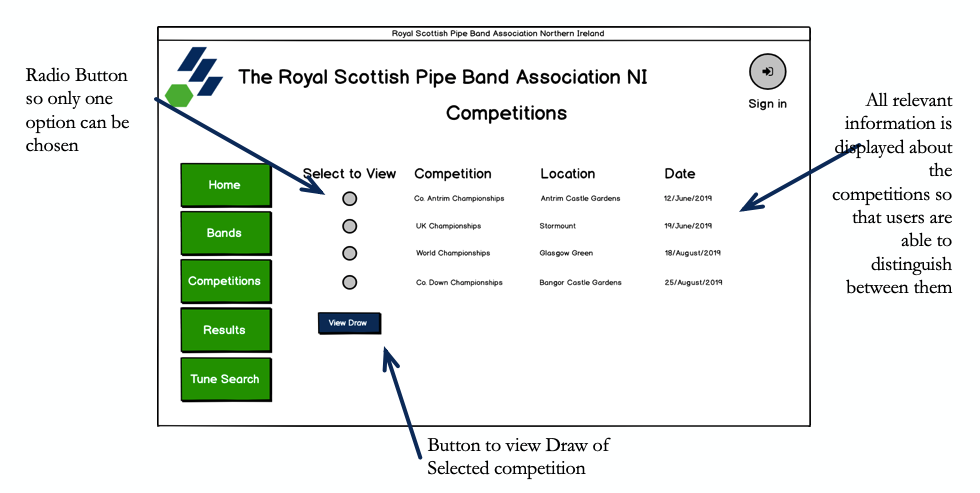
Whether they have been filtered or not, all the bands that are present in the table will be organised in alphabetical order. Each box will have a label at the top to explain the data in the field.

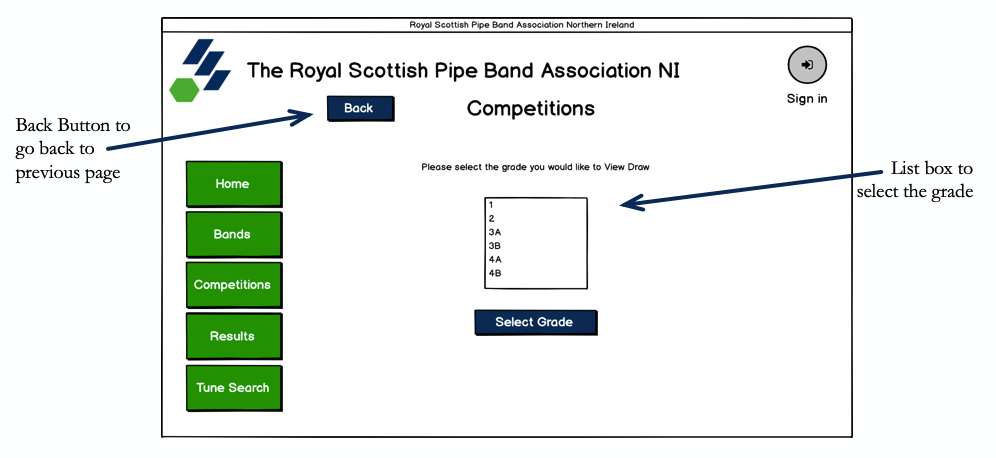


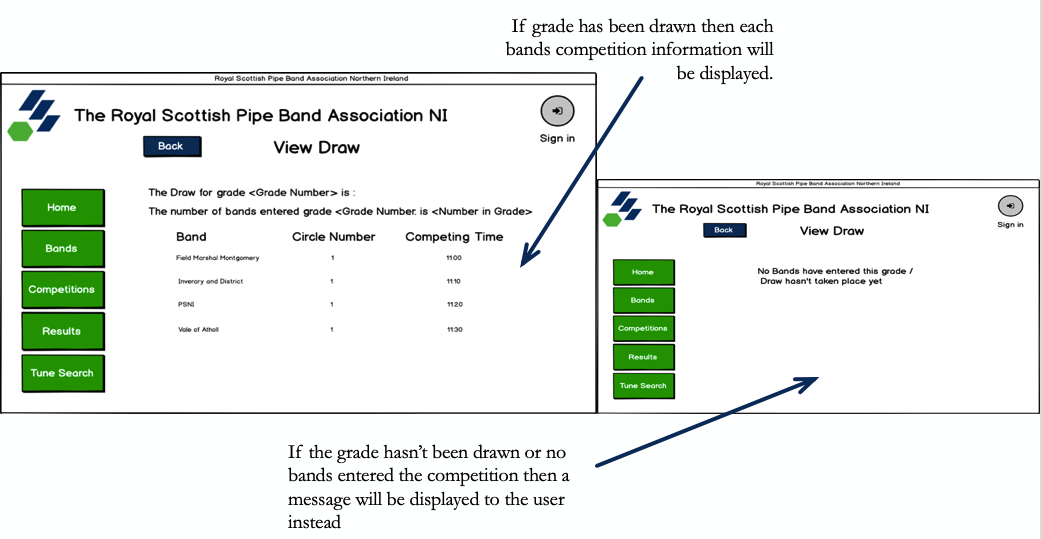
|  |  |
| --- | --- |
| Pseudocode to display all the bands on the page | START  NameListbox = ListBox()  GradeListbox = ListBox()  BranchListbox = ListBox()  AddressListbox = ListBox()  TimeListbox = ListBox()  TartanListbox = ListBox()  AllBands = SELECT BandName, BandGrade, BandBranch, HallName, AddressLine1, AddressLine2, County, Postcode, PracticeTime, Tartan FROM BandAccount  FOR Band IN AllBands  BEGIN  Insert Band[0] into NameListBox  Insert Band[1] into GradeListBox  Insert Band[2] into BranchListBox  Insert concatenate(Band[3] + Band[4] + Band[]5 + Band[6] + Band[7]) into AddressListBox  Insert Band[8] into TimeListBox  Insert Band[9] into TartanListBox  NEXT  END |
| Pseudocode for searching by grade filter | START  NameListbox = ListBox()  GradeListbox = ListBox()  BranchListbox = ListBox()  AddressListbox = ListBox()  TimeListbox = ListBox()  TartanListbox = ListBox()  Grade = Get(GradeSelected from RadioButton)  BandsInFilter = SELECT BandName, BandGrade, BandBranch, HallName, AddressLine1, AddressLine2, County, Postcode, PracticeTime, Tartan FROM BandAccount WHERE BandGrade = Grade  FOR Band IN BandsInFilter  BEGIN  Insert Band[0] into NameListBox  Insert Band[1] into GradeListBox  Insert Band[2] into BranchListBox  Insert concatenate(Band[3] + Band[4] + Band[]5 + Band[6] + Band[7]) into AddressListBox  Insert Band[8] into TimeListBox  Insert Band[9] into TartanListBox  NEXT  END |
| Pseudocode for searching by band name filter | START  NameListbox = ListBox()  GradeListbox = ListBox()  BranchListbox = ListBox()  AddressListbox = ListBox()  TimeListbox = ListBox()  TartanListbox = ListBox()  INPUT BandNameEntered  BandsInFilter = SELECT BandName, BandGrade, BandBranch, HallName, AddressLine1, AddressLine2, County, Postcode, PracticeTime, Tartan FROM BandAccount WHERE BandName = BandNameEntered  FOR Band IN BandsInFilter  BEGIN  Insert Band[0] into NameListBox  Insert Band[1] into GradeListBox  Insert Band[2] into BranchListBox  Insert concatenate(Band[3] + Band[4] + Band[5] + Band[6] + Band[7]) into AddressListBox  Insert Band[8] into TimeListBox  Insert Band[9] into TartanListBox  NEXT  END |

**“All users have the ability to view and search for a competition on the competitions page.”**

This screen is will also be available to all users of my application. This Competitions page will display all the competitions of the season. Competitions displayed in alphabetical order. This page allows the user to select a competition to view the Draw for each grade using the radio button selection on the page. They must select the grade they would like to see the draw for then the circle number and competing time will be displayed if the competition has been drawn.

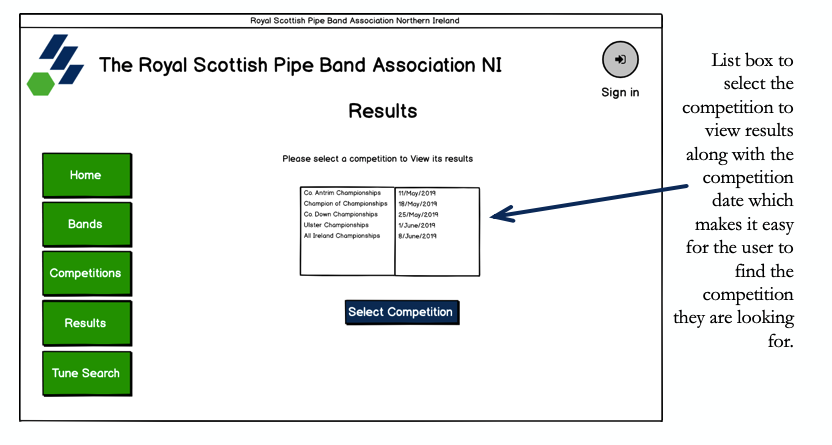


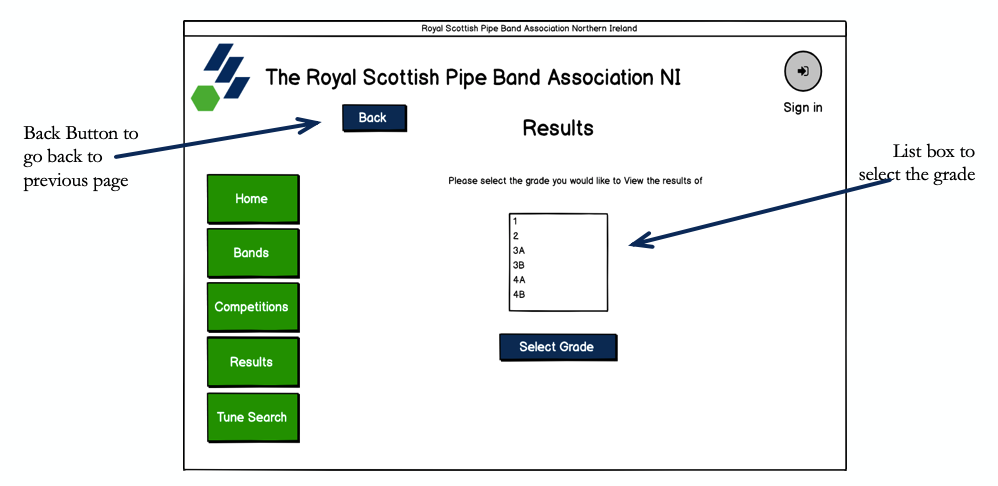


****

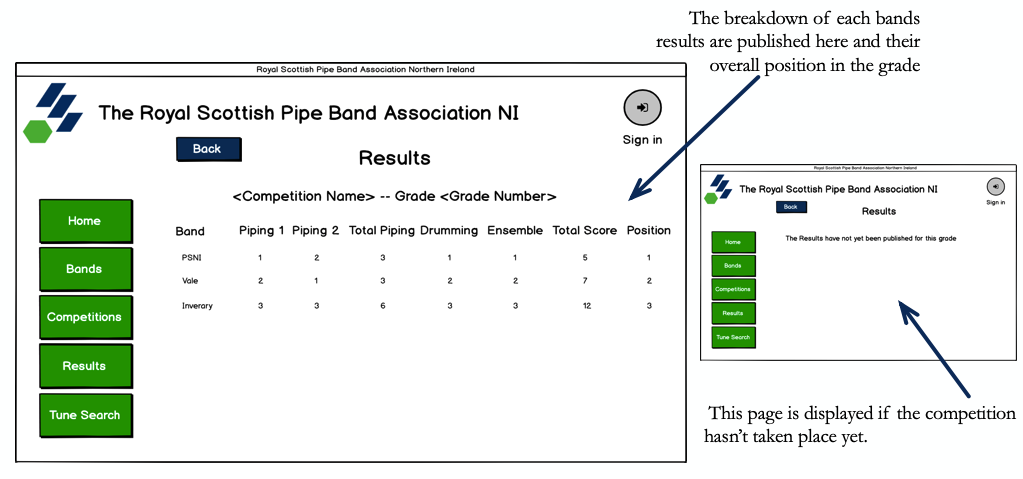
|  |  |
| --- | --- |
| Viewing a competition draw – This is achieved by the user selecting the competition and then followed by the grade. The Database will be queried with this retrieved data and display all the bands in the grade along with the circle number and competing time they have been given. | START  CompetitionIDSelected = Get(CompetitonSelected)  GradeSelected = Get(GradeSelected)  BandDrawQuery = SELECT BandID, BandName FROM BandAccount WHERE Grade = GradeSelected  FOR Band IN BandDrawQuery  BEGIN  DrawQuery = SELECT CircleNum, CompetingTime FROM CompetitionDraw . WHERE BandID = Band[0]    IF results from DrawQuery THEN  Display (BandName, CirlceNumber, CompetitionTime)  ELSE  Display(“No Bands have entered this grade or Draw hasn’t taken place”)  NEXT  END |

**“All users have the ability to view and search for a grades’ result in a competition on the results page.”**

As before, this page will be available to all users of my system. I have decided to again use list boxes to implement this objective to meet its criteria. The user is prompted with some text above the list box explaining to them that they must select a competition to view its results.

Once selected they must go through the same procedure and select the grade they wish to view.

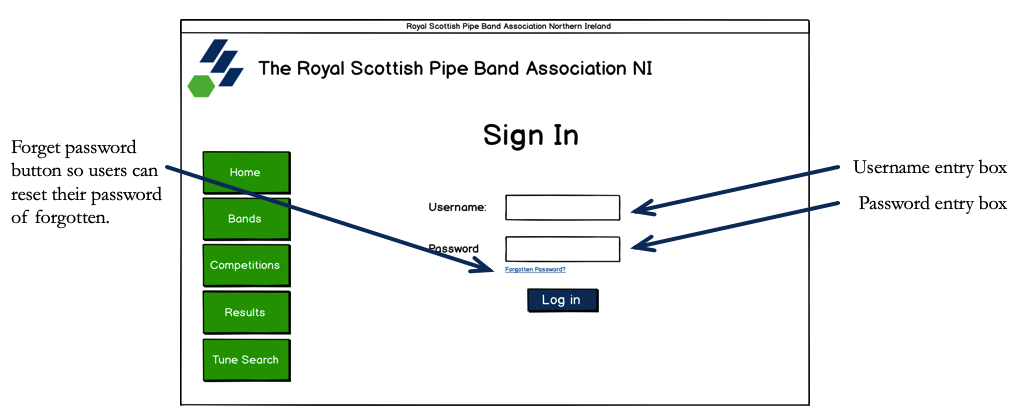
Once they have selected the grade, they will be presented with the results in the grade including each sections score, which are all aligned on the screen to make it easy to read. If the competition has not taken place and therefore results not published, instead a message displayed telling the user the reason the results cannot be displayed.



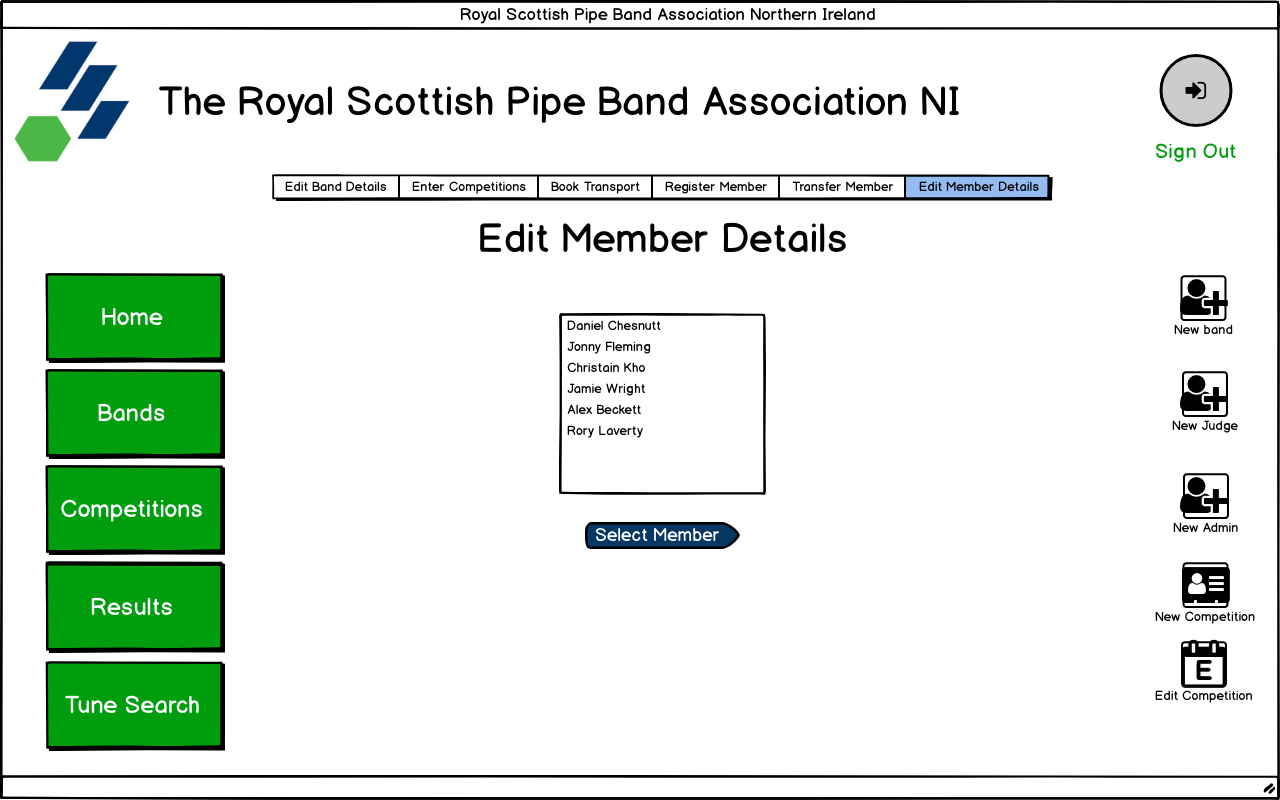
|  |  |
| --- | --- |
| Displaying Results to user – Before the results can be displayed the user must select the competition and grade they want to view the results for. Once this is done. The Database is queried so that only the results of that grade of the competition are returned to the user. This will be done by getting all the bands entered into the grade of the competition and ordering them by their final position in the grade, eg ranked best to worst, then displayed. | START  CompetitionSelected = Get(CompetitionSelection)  GradeSelected = Get(GradeSelection)  BandsInGradeSelected = []  GradeResults = []  BandsInCompetition = SELECT BandID FROM BandsEntered WHERE CompetitionID = CompetitionSelected  FOR Band in BandsInCompetition  BEGIN  BandGrade = SELECT Grade FROM BandAccount WHERE BandID = Band  IF BandGrade == GradeSelected THEN  BandsInGradeSelected.append(Band)  NEXT  FOR BandID in BandsInGradeSelected  ResultsBandsInGrade = SELECT Piping1Score, Piping2Score, TotalPiping,  DrummingScore, EnsembleScore, TotalScore, Position WHERE BandID = BandID  AND CompetitionID = CompetitionSelected  GradeResults.append(ResultsBandsInGrade)  NEXT  Sort by Position ASCENDING (GradeResults)  Display(Results)  END |

**“All users have the ability to login if they have an account.”**

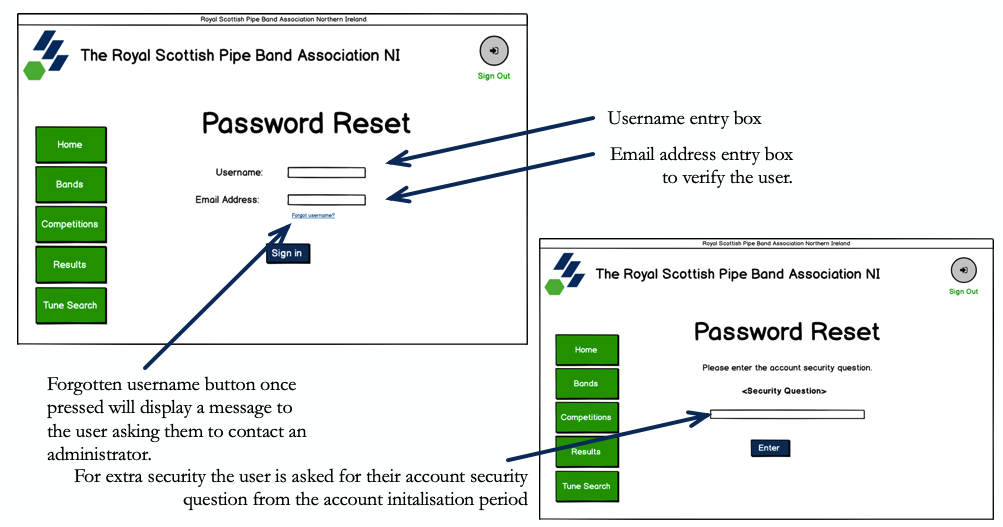
When the user presses the log in button on the top right corner of their screen, they will be provided this screen which will allow them to log onto to their account using their credentials. This page consists of only text entry boxes that are supported by reflective labels making the reading and layout of this form simple and intuitive. The forget password button is small and not as noticeable as the blue button to be pressed to log in.

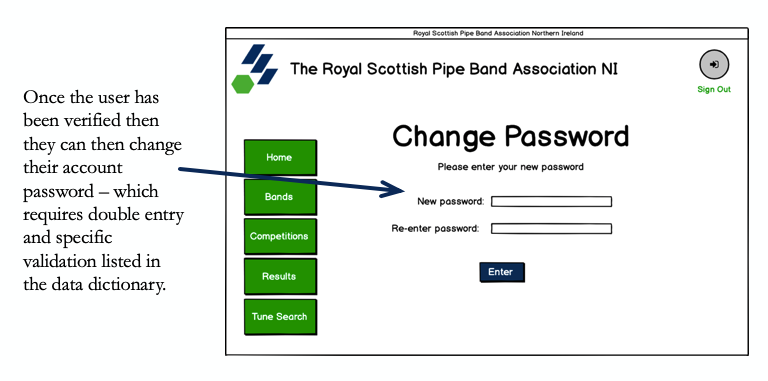


|  |  |
| --- | --- |
| Logging In – Users must enter their username and password into the respective input fields. When they press log in their details are taken and checked to see if they exist and in which table (Band, Judge or admin) this will determine which access levels they will receive. | START  Username = Get(Username)  Password = Get(Password)  CheckBandAcc = SELECT FirstName FROM BandAccount WHERE Username = Username AND Password = Password  IF CheckBandAcc THEN  OUTPUT (“Welcome” + str(CheckBandAcc))  Display(BandFeatures)  ELSE:  CheckJudgeAcc = SELECT FirstName FROM JudgeAccount WHERE Username = Username AND Password = Password  IF CheckJudgeAcc THEN  OUTPUT (“Welcome” + str(CheckJudgeAcc))  Display(JudgeFeatures)  ELSE:  CheckAdminAcc = SELECT FirstName FROM AdminAccount WHERE Username = Username AND Password = Password  IF CheckAdminAcc THEN  OUTPUT (“Welcome” + str(CheckAdminAcc))  Display(BandFeatures)  Display(JudgeFeatures)  Display(AdminFeatures)  ELSE:  OUTPUT(“No account found please ensure your credentials are correct”)  END |



**“All Account holders have the ability to reset their login credentials.”**

The user of this system will be brought to this page if they have forgotten their password and have pressed the forgot password button on the login in screen as seen in the objective above. They are prompted like before to enter their username and email address this time. This is extra verification of the user to ensure they are who they are saying they are. Again, the same as before if they have forgotten their username there is a smaller button to press which will inform the user that they must contact an administrator to get access to their account again.

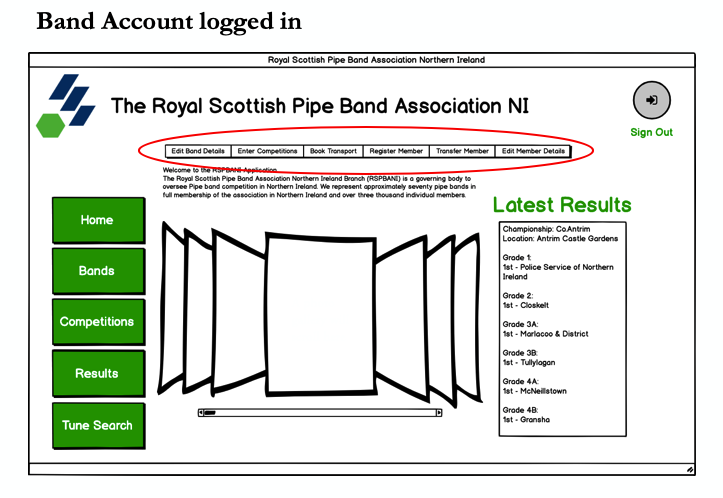


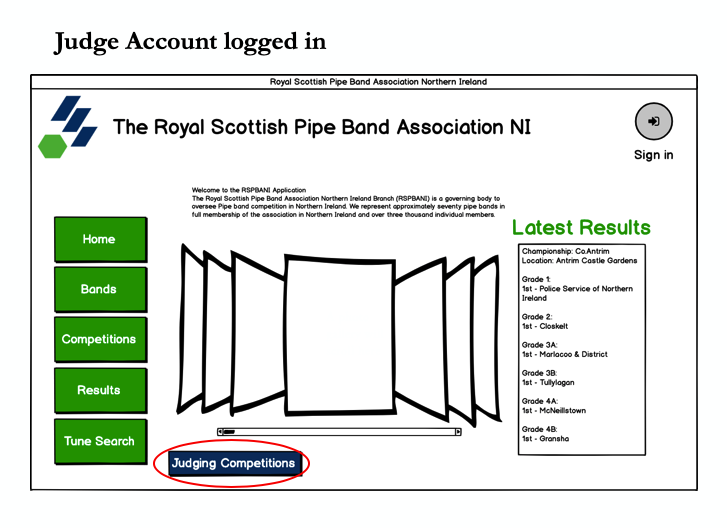
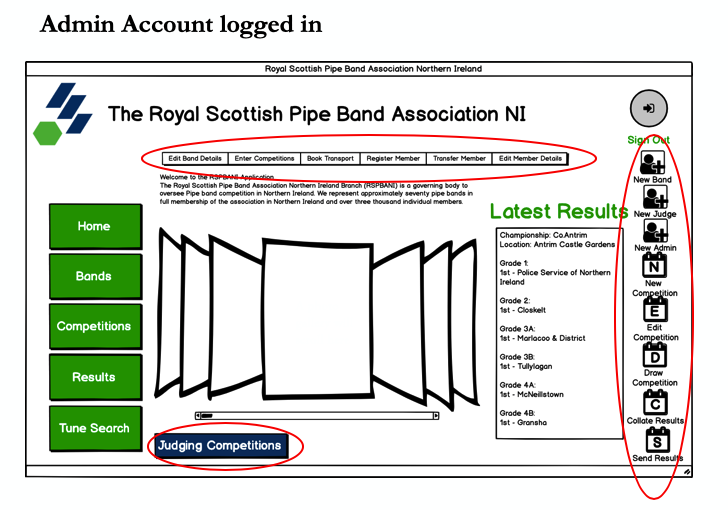
|  |  |
| --- | --- |
| Password reset - | START  FUNCTION on button press  Username = Get(Username)  EmailAddress = Get(EmailAddress)  AdminCheck = SELECT SecurityQ, SecurityQAns FROM AdminAccount  IF AdminCheck  SecurityQuestion = SecurityQ  SecurityQAnswer = SecurityQAns  DISPLAY(AskSecurityQ)  ELSE:  BandCheck = SELECT SecurityQ, SecurityQAns FROM BandAccount  IF BandCheck  SecurityQuestion = SecurityQ  SecurityQAnswer = SecurityQAns  DISPLAY(AskSecurityQ)  JudgeCheck = SELECT SecurityQ, SecurityQAns FROM JudgeAccount  IF JudgeCheck  SecurityQuestion = SecurityQ  SecurityQAnswer = SecurityQAns  DISPLAY(AskSecurityQ)  ELSE:  OUTPUT(“No account founded matching these credentials”)  FUNCTION AskSecurityQ  Populate interface with SecurityQuestion  Populate entry box for their SecurityQuestion Answer  FUNCTION on button press  InputSecurityQAnswer = Get(SecurityQuestionAnswer)  IF InputSecurityQAnswer == SecurityQAnswer THEN  DISPLAY(ChangePasswordFrame)  ELSE:  OUTPUT(“Security Question Answer does not match”)  FUNCTION ChangePassworFrame  Populate Frame with password label and entry box  Populate Frame with re-enter password label and entry box  FUNCTION on button press  NewPassword = Get(Password)  NewPasswordDouble = Get(Re-enterPassword)  IF CheckPasswordValidation(NewPassword) \* == True THEN  IF NewPassword == NewPasswordDouble THEN  UPDATE Database (Password)  ELSE:  OUTPUT(“Passwords do not match”)  DISPLAY(ChangePasswordFrame)  ELSE:  OUTPUT(“Password Invalid”)  DISPLAY(ChangePasswordFrame)  END |

\*Note CheckPasswordValidation is a validation routine that has been outlined in the validation section at the end of this section

**“Different permission levels for band, judge and admin accounts once logged in.”**

Once logged in these are the different access levels that are available and the additional features that come with the particular levels are circled in red. The designs below show the positioning of these buttons for each user.

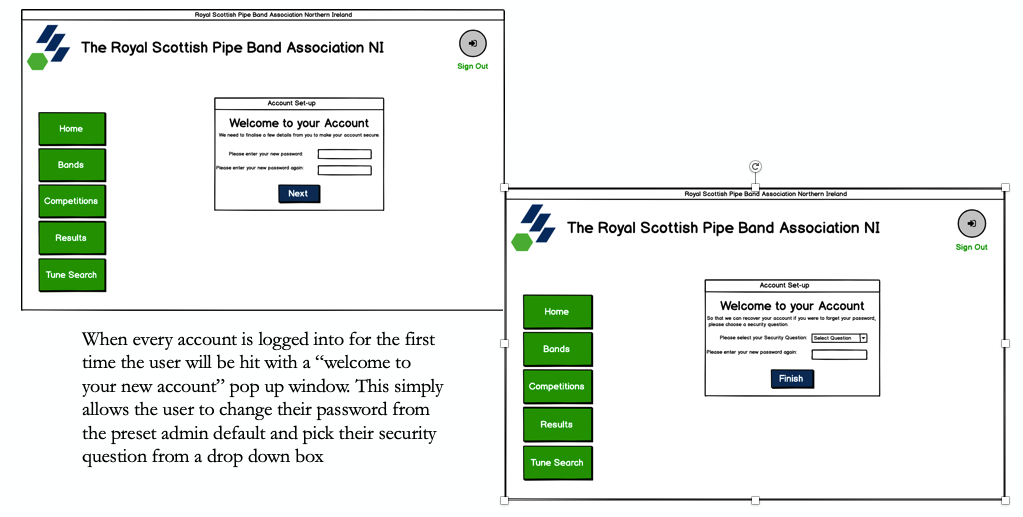




*See Pseudocode for Log In with access levels in the Login objective above.*

**“Update and set password, security question and answer when accounts are logged in for the first time.”**

When the user enters their password. Both fields will be checked to ensure they match and then checked to ensure they have met the required strength. This is documented in the validation processes found later in this document. On the button press, if password is acceptable then this new password will be updated in the database and the current password is overridden otherwise the user will be prompted to re-enter their new password.



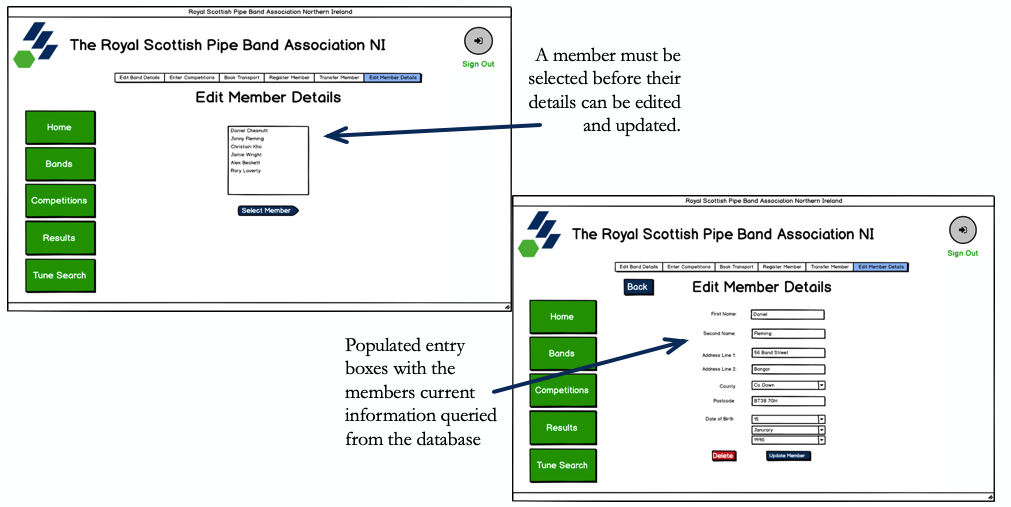
The user will then be presented with a new page to select their security question and answer. Both fields will be saved to the database in the record belonging to the logged in user. The program will check to ensure the user has selected a question and has provided an answer. On button press, if all data is valid, the data will be saved, and user will be brought to the home screen as pop up window will be destroyed.

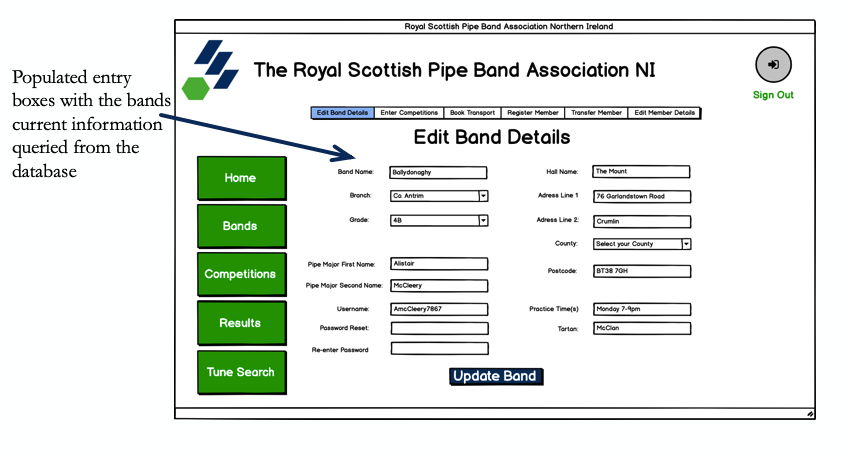
Processing stages

|  |  |
| --- | --- |
| Checking for password and double entry is a match | START  On “Next” button press  Get passwords from entry boxes  PasswordCheck = False  If CheckPassword (Password1) == True:  If password1 == password2 :  PasswordCheck = True  If passwordCheck == True:  UPDATE BandAccount SET Password = Password1 WHERE BandID = BandLoggedinID  Display Security Question screen  ELSE:  Display Password frame to ask user to enter password again  END |
| Checking for security question selected and answer given | START  On “finish” button press  Get security question and answer from entry boxes  ValidQuestion = False  ValidAnswer = False  IF SecurityQuestion != “Select a Question”:  ValidQuestion = True  If SQAnswer = “” or “ “:  OUTPUT(“Please enter your answer to the selected question”)  ELSE:  ValidAnswer = True  ELSE:  OUTPUT (“Please select a question”)  IF ValidAnswer == True AND ValidQuestion == True:  UPDATE BandAccount SET SecurityQuestion = SecurityQuestion , SecuirtyQAns = WHERE BandID = BandLoggedinID  Destroy pop up window  Display Home page  ELSE:  Return to Security Question Frame  END |

**“Band accounts will have the ability to update the Bands details & their band members details.”**

This object allows for a pipe major of a band who has logged in or an admin to update and edit the details of the band or the details of the members. Once saved all data will be revalidated to ensure all new data entered is valid. If all data is valid then update the bands record to hold all the information that is now found in these entry boxes.



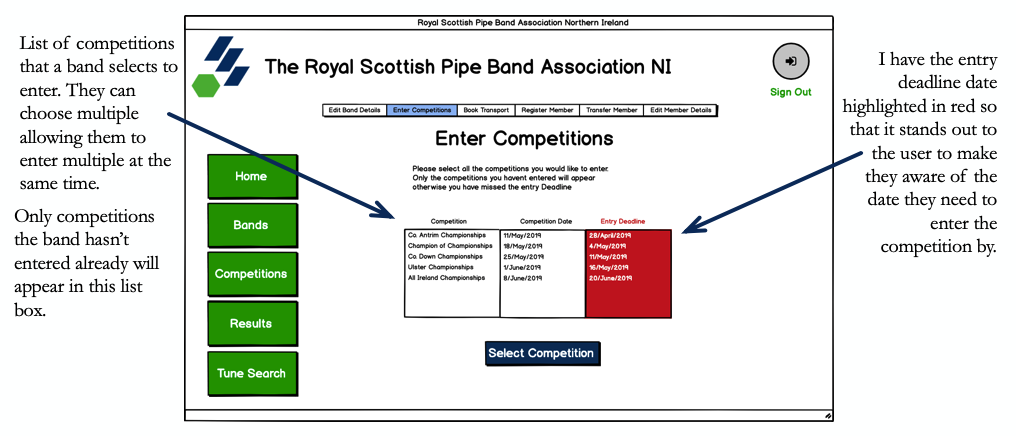


Processing stages

|  |  |
| --- | --- |
| Edit Band Details | START  BandNameCheck = False  BandBranchCheck = False  BandGradeCheck = False  PMFirstNameCheck = False  PMSecondNameCheck = False  CheckUsername = False  AdressLine1Check = False  CountyCheck = False  PostcodeCheck = False  BandDetails = SELECT \* FROM BandAccount WHERE BandID = BandIDofBandLoggedIn  Populate the frame was seen in the design above  Populate entry boxes with respectful fields that have been retrieved from database  On button click  IF UsernameCheck(Username.get()) == True:  CheckUsername = True  IF CheckAlphaAndDigit(BandName.get()) == True:  BandNameCheck = True  IF BandBranch != “Select Branch”:  BandBranchCheck = True  IF Grade != “Select Gade”:  GradeCheck = True  IF AllAlpha(PMFirstName.get()) == True:  PMFirstNameCheck = True  IF AllAlpha(PMSecondName.get()) == True:  PMSecondNameCheck = True  IF CheckAlphaAndDigit (AddressLine1.get()) == True:  AddressLin1Check = True  IF County != “Select County”:  CountyCheck = True  IF CheckPostcode(Postcode.get()) == True:  PostcodeCheck = True  IF PostcodeCheck == True AND CountyCheck == True AND BandNameCheck == True AND BandBranchCheck == True AND BandGradeCheck == True AND PMFirstNameCheck == True AND PMSecondNameCheck == True AND CheckUsername == True AND AdressLine1Check == True:  UPDATE BandAccount SET \* = BandDetails WHERE BandID = BandIDofLoggedIn  ELSE:  OUTPUT(X Field not valid)  END |
| Edit Member Details | See pseudocode from edit band details above. The processing stages are the same however the fields and data to be populated will be queried from Members table instead. Validation routine used will be the same and can be found below. |
| Delete Member | START  MemberSelected = Selected member from listbox  On “Delete” Button click  DELETE FROM MemberDetails WHERE MemberID = MemberSelected  END |

**“Bands will have the ability to enter competitions”**

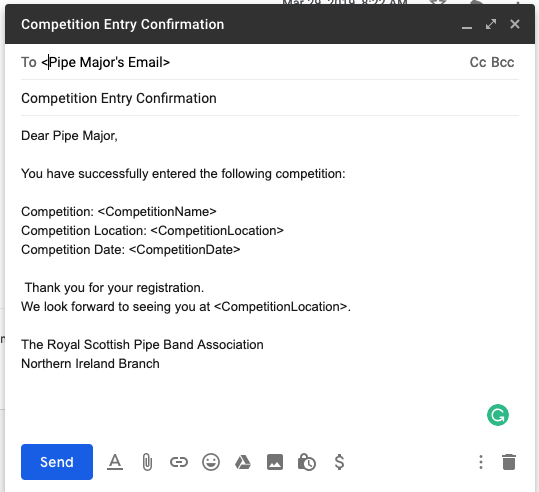
In this feature, bands can enter by selecting the competition and pressing enter. The list box displayed on this page will only contain competitions that the band has not already entered, and the competition deadline has not passed (i.e. the competition has not been drawn). The pipe Major will receive an email as confirmation of entering the competition.



Processing stages

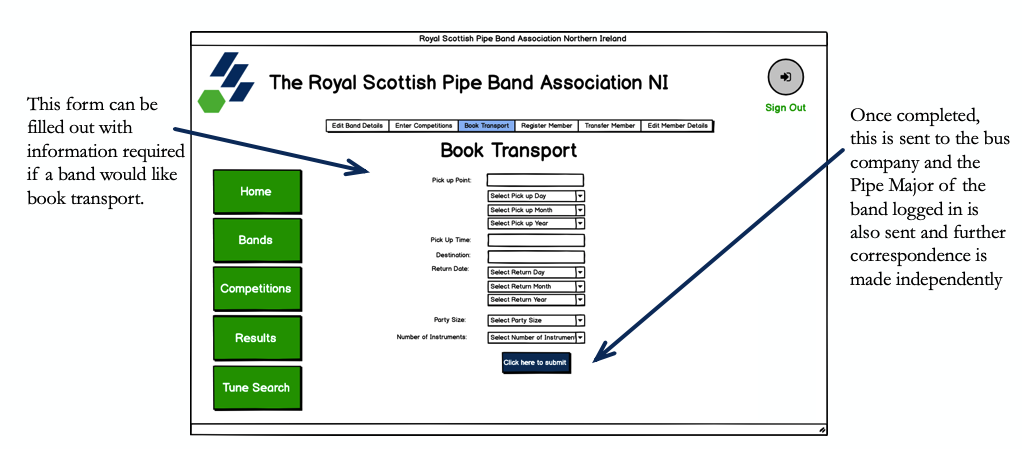
|  |  |
| --- | --- |
| Displaying competitions that are Displayed in list box | START  CompsNotDrawn = SELECT CompetitionID FROM Competitios WHERE Drawn = False  CompetitionsEntered = SELECT CompetitionID FROM BandsEntered WHERE BandID = BandIDofLoggedIn  CompsToBeDisplayed = Difference (CompsNotDrawn, CompetitionsEntered)  Populate Listbox with CompsToBeDisplayed  END |
| Entering Competitions | START  CompetitionSelected = CompetitionSelected from listbox  INSERT INTO BandsEntered (BandID, CompetitionID, DateEntered) Values (BandIDofLoggedIn, CompetitionSelected, CurrentDate)  END |

Output:

The email that will be sent to the pipe major, will include all the details about the competition. These details will be queried from the database and retrieved using the Competition ID of the selected competition in the list box. If multiple are selected. There will be an email for each entry.

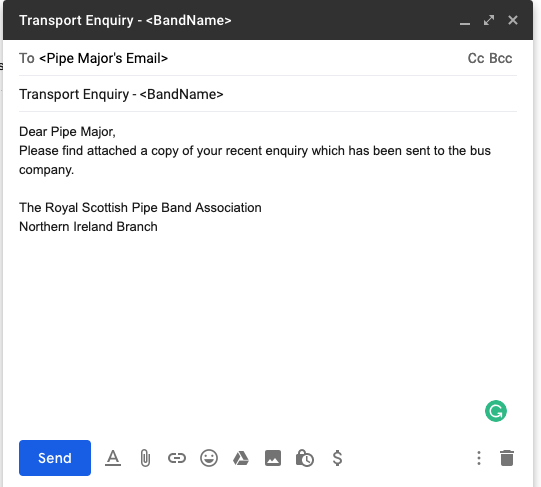
**“Bands have the option of booking transport.”**

This objective allows for bands to enquire about transport. From my desk-based research, I have included the following fields to ensure the bus company receive enough information to provide a quote to the band.

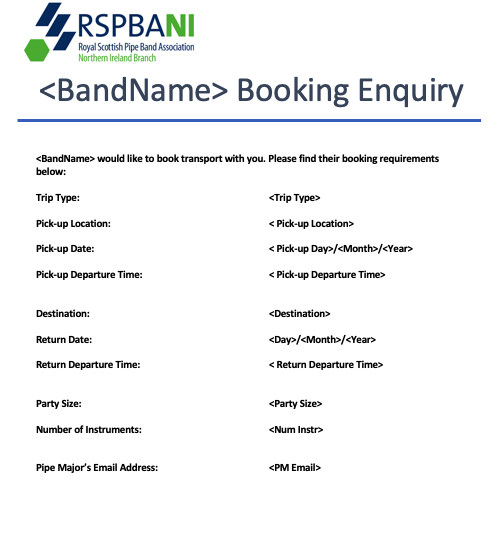


Processing stages

|  |  |
| --- | --- |
| Booking Transport | Display Frame as shown above in interface design.  PickUpPointCheck = False  PickUpDayCheck = False  PickUpMonthCheck = False  PickUpYearCheck = False  PickUpTimeCheck = False  DestintationCheck = False  ReturnDayCheck = False  ReturnMonthCheck = False  ReturnYearCheck = False  ReturnTimeCheck = False  PartySizeCheck = False  NumInstrCheck = False  Check  IF PickUpPoint.get() != “”:  PickUpPointCheck = True  IF PickUpDay.get() != “Select Day”:  PickUpDayCheck = True  IF PickUpMonth.get() != “Select Month”:  PickUpMonthCheck = True  IF PickUpYear.get() != “Select Year”:  PickUpYearCheck = True  IF PickUpTime.get() != “”:  PickUpTimeCheck = True  IF Destination.get() != “”:  DestinationCheck = True  IF ReturnTime.get() != “”:  ReturnTimeCheck = True  IF ReturnDay.get() != “Select Day”:  ReturnDayCheck = True  IF ReturnMonth.get() != “Select Month”:  ReturnMonthCheck = True  IF ReturnYear.get() != “Select Year”:  ReturnYearCheck = True  IF Int(PartySize.get()) <= 150:  PartySizeCheck = True  IF int(NumInstr.get()) < 100:  NumInstrCheck = True  IF PickUpPointCheck == True AND PickUpMonthCheck == True AND PickUpYearCheck == True AND PickUpTimeCheck == True AND DestinationCheck == True AND ReturnTimeCheck == True AND ReturnDayCheck == True AND ReturnMonthCheck == True AND ReturnYearCheck == True AND PartySizeCheck = True AND NumInstrCheck ==True:  Email = SELECT EmailAddress From BandAccount WHERE BandID = BandLoggedIn.  Create Document and add all these fields along with Band details and Pipe Majors Email Address - as seen in the designed output below.  Email Pipe Major(output shown in output section) this will include this document attached as confirmation of their enquiry.  Email Bus company (output shown in output section) this will include this document attached.  ELSE:  OUTPUT (X Field not valid)  END |

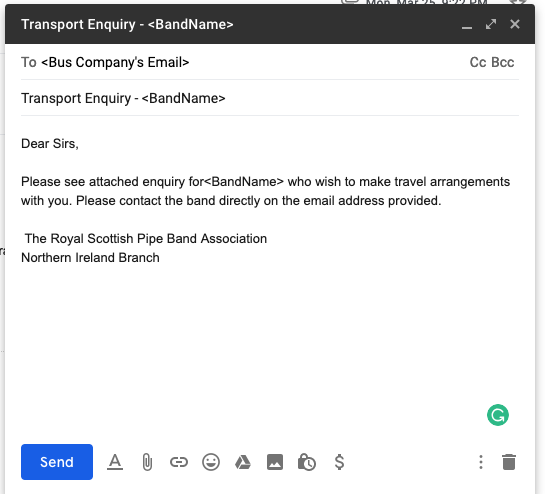
**Outputs:**

This is the mock-up of the email sent to the Pipe Major on enquiring transport using the form. The details will be attached to keep as a record of their request.



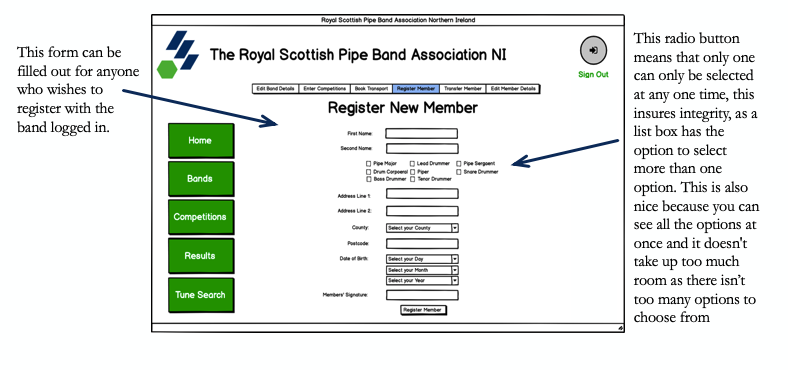
This designed output will be produced with the information that the Pipe Major has entered into the Booking Enquiry Form.

The respected Fields will be laid out as shown. I have designed this document in such a way that all the information with regards to pick up, return and contact details are split making it easier to read.

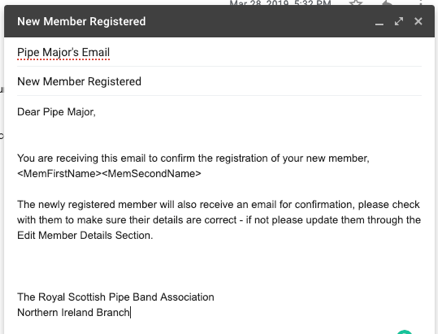
Similar to the Email sent to the Pipe Major, this is the email which the bus company can expect to receive from the Association when they receive an enquiry.

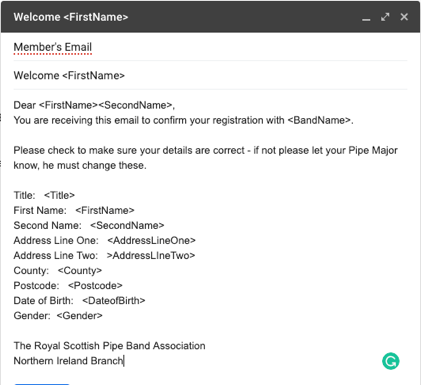
Along with this email they will find attached the document with the Bands requirements.

**“Bands have the ability to register new players and transfer existing members.”**

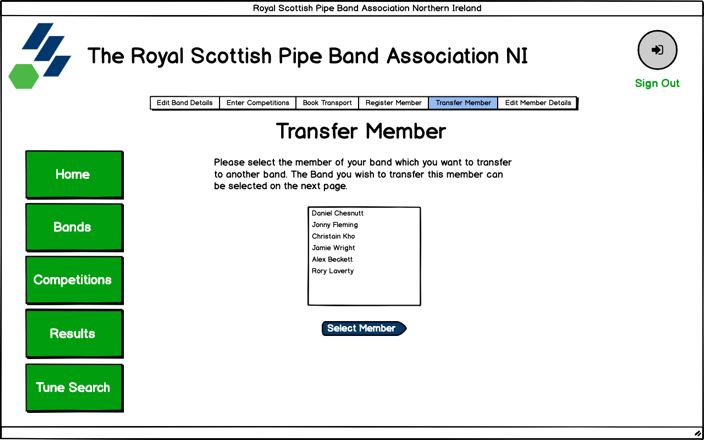
The Pipe Major has the ability to register new members to their band. They must fill out the following form. Once they press the register button, this information is stored, and a new record is made for this new member in the MembersDetails Table within the database. The Member is linked to the registered band through a foreign key of the BandID from the BandAccount Table in the Members Details Table. This exists for each member in the table. The Pipe Major receives an email of confirmation. The Member will also receive an email, their details will be sent over email as so they can ensure their details have been stored accurately and fairly.

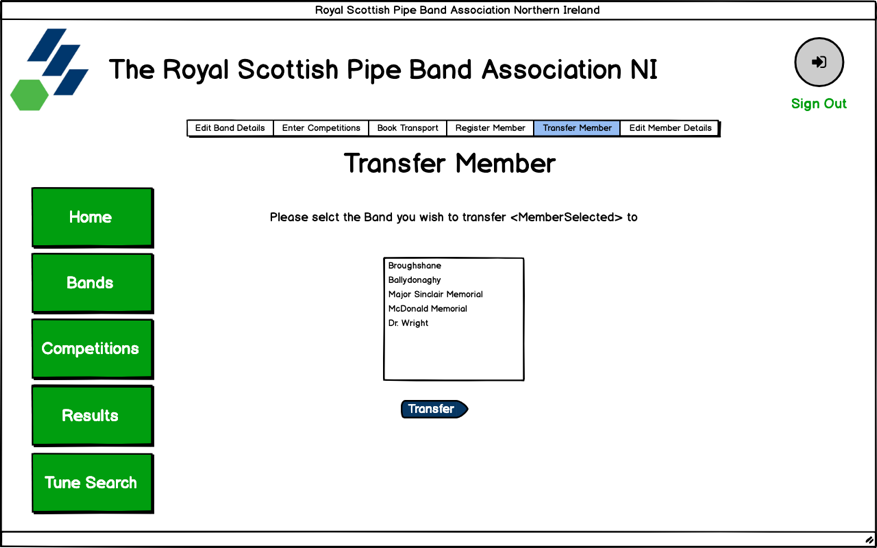
Outputs:

This output shows a mock-up of the email the Pipe Major of the band, that has registered a new member. The name of the Member will be queried from the form and used in the email. I have decided to include this as so the email is specifically being generated for each new member and not a generic email.

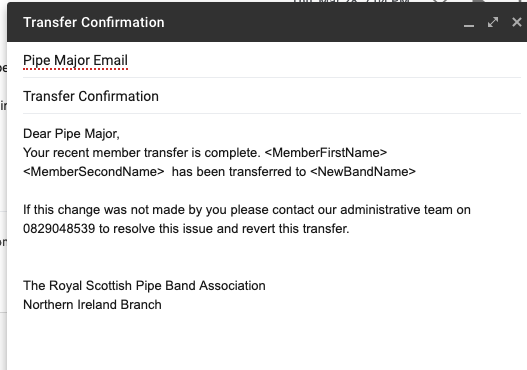
This output shows the mock-up of the email received by the newly registered member. It includes their details, so they can ensure they have been entered correctly by the Pipe Major.

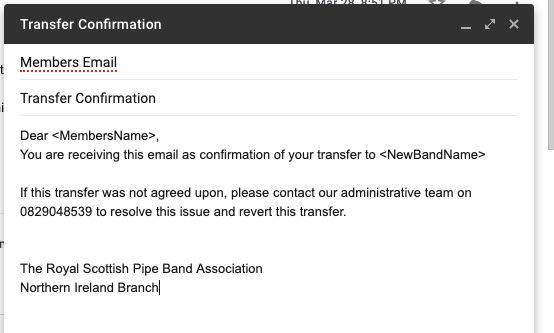
When a Pipe Major wishes to transfer a member to another band on the members request. The Pipe Major must select the member they wish to transfer and then to which band once they have selected the member. I have designed these using list boxes. This ensures they PM can only select one member and one band only.





Output:

Again, this mock-up shows the email that must be sent to the pipe major on transferring a member. This email will include the Members Name and the band which they have been transferred to.

The transferred member will receive this email on transfer between bands. The purpose of this email is to let the member know that they have been transferred to the named band.

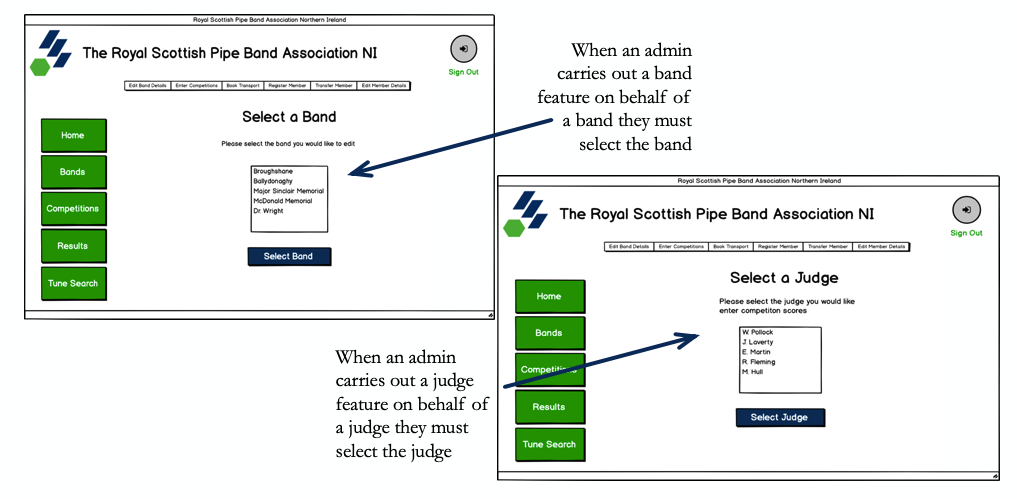
Processing stages

|  |  |
| --- | --- |
| Registering Member | See pseudocode for registering a new band below. This feature instead will save the members details to the MembersDetails table. |
| Transferring Member | START  SelectedMember = Selected Member from list box  CurrentBand = BandIDofLoggedIn  ListOfBands = SELECT BandID FROM BandAccount  ListOfBands.remove(CurrentBand)  Display new page and populate another list box with ListOfBands.  SelectedBand = Selected Band from list box  Ask user are they sure they want to transfer <SelectedMember> to <SelectedBand>  UPDATE BandMembers SET BandID = SelectedBand WHERE MemberID = SelectedMember  Email BandMember, New Band, and old band for confirmation  END |

**“Admins have the ability to carry out these band features above on behalf of the band.”**

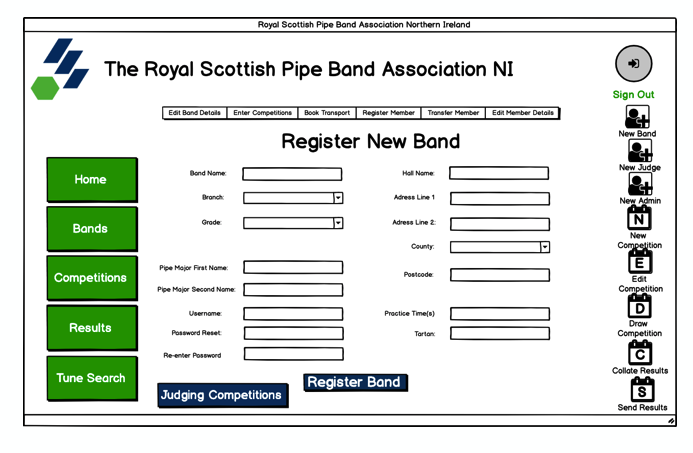
This feature allows admins to make the necessary changes to any band. This feature will work by instead of using the Band ID of the Band which has been logged in their ID of the Band the Admin has selected will be used throughout this feature.

The same logic is applied to the admin when they wish to judge a competition. The ID of the judge selected will be used instead of the ID of the Judge logged on.

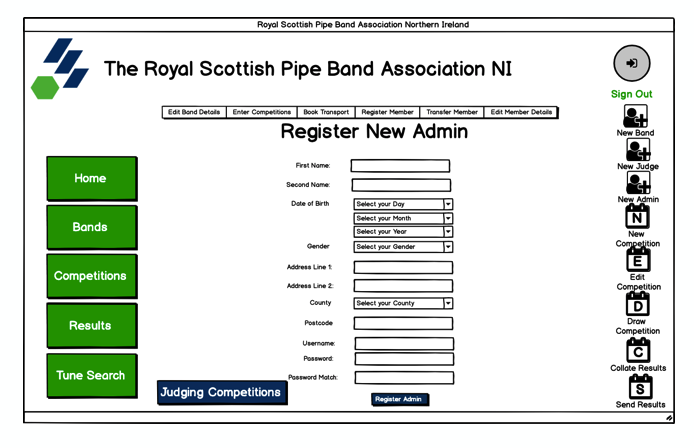
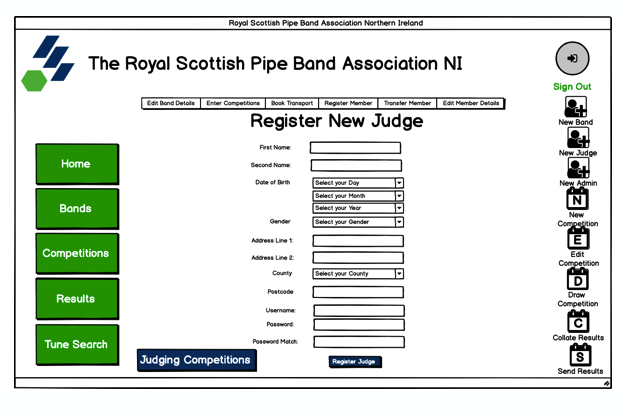


**“Admins have the ability to add and remove band accounts, judge accounts and admin accounts. When a band account is created an email is sent to the Pipe Major to include the initial log in details”**

I have designed this feature to make it simple for the user to use. I have used a combination of drop-down boxes and entry boxes to make it easier to enter the correct data. The use of drop-down boxes allows the user to only select a valid option. This means that the data selected can be referred to something else without ambiguity.

In each case of registration all the data is pulled from the entry boxes and drop-down lists and checked to ensure it is valid. If all data isn’t valid a message box will alert the user – Form is returned in same state to allow user to correct mistakes without losing the rest of their inputted information.

When a band, admin or judge is registered, their information is stored in the BandAccount, AdminAccount or JudgeAccount respectively.

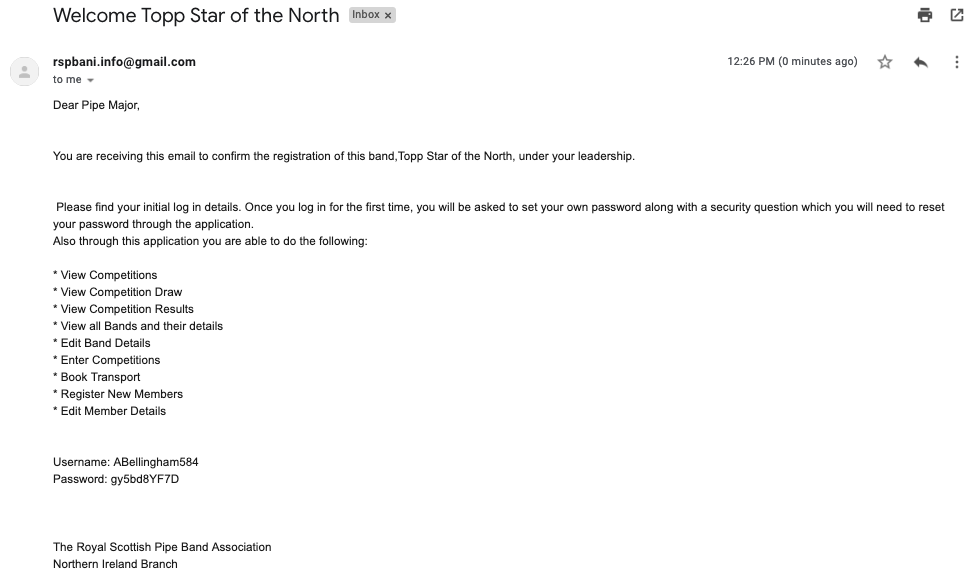


Processing stages

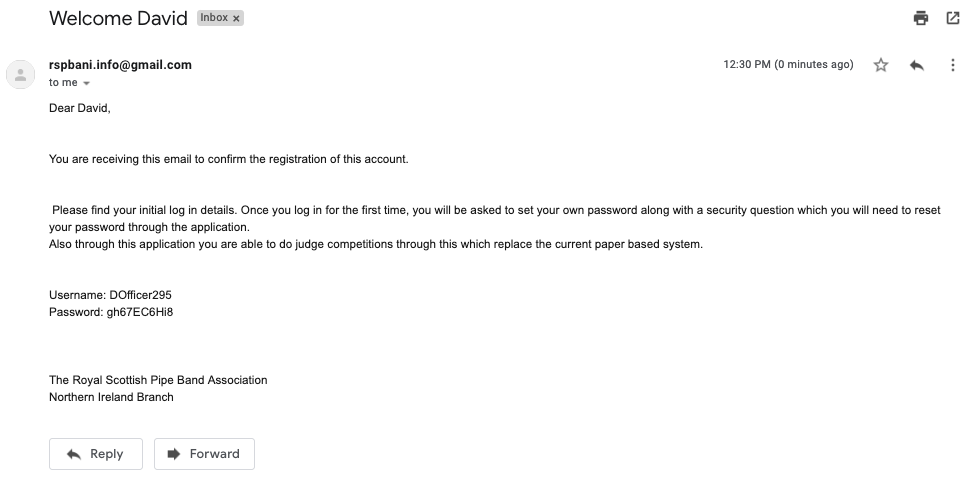
|  |  |
| --- | --- |
| Registering a new Band | START  Populate Frame as seen in interface design.  On “Register Band” Button click  BandNameCheck = False  BandBranchCheck = False  BandGradeCheck = False  PMFirstNameCheck = False  PMSecondNameCheck = False  CheckUsername = False  PasswordCheck = False  AdressLine1Check = False  CountyCheck = False  PostcodeCheck = False  IF UsernameCheck(Username.get()) == True:  CheckUsername = True  IF PasswordCheck(Password.get() == True  PasswordCheck = True  IF CheckAlphaAndDigit(BandName.get()) == True:  BandNameCheck = True  IF BandBranch != “Select Branch”:  BandBranchCheck = True  IF Grade != “Select Gade”:  GradeCheck = True  IF AllAlpha(PMFirstName.get()) == True:  PMFirstNameCheck = True  IF AllAlpha(PMSecondName.get()) == True:  PMSecondNameCheck = True  IF CheckAlphaAndDigit (AddressLine1.get()) == True:  AddressLin1Check = True  IF County != “Select County”:  CountyCheck = True  IF CheckPostcode(Postcode.get()) == True:  PostcodeCheck = True  IF PostcodeCheck == True AND CountyCheck == True AND BandNameCheck == True AND BandBranchCheck == True AND BandGradeCheck == True AND PMFirstNameCheck == True AND PMSecondNameCheck == True AND CheckUsername == True AND AdressLine1Check == True AND PasswordCheck = True:  INSERT INTO BandAccount (BandName, BandBranch, Grade, PMFirstName, PMSecondName, HallName, AddressLine1, AddressLine2, County, Postcode, EmailAddress, Username, Password)  Email Pipe Major(output shown in output section) this will include username and initial password.  ELSE:  OUTPUT(X Field not valid)  END |
| Registering a new Admin | See pseudocode for registering a new band above instead of inserting into Band Account information will be committed to the AdminAccount Table |
| Registering a new Judge | See pseudocode for registering a new band above instead of inserting into Band Account information will be committed to the JudgeAccount Table |

Outputs of the system:

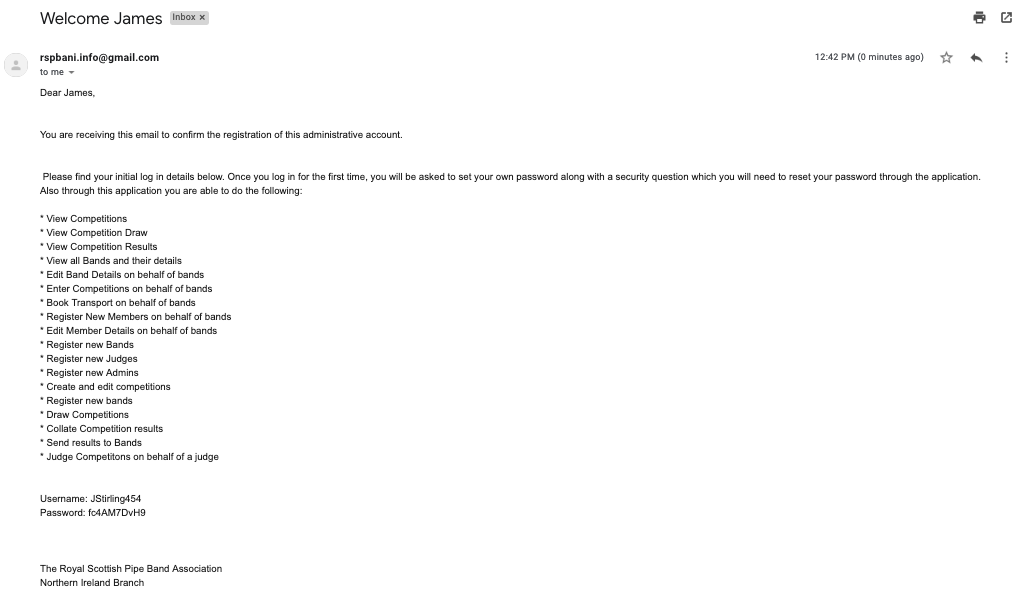
I have included a mock-up of the email that the pipe major of the newly registered band will receive which will include their initial log in credentials to allow the user pipe major to log in to the system.



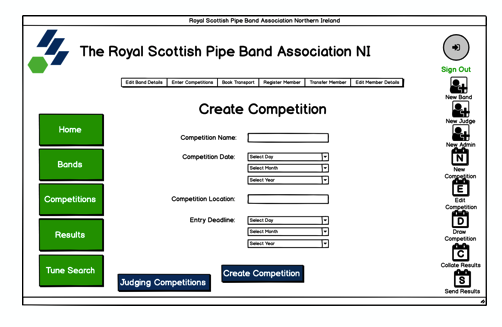
Similar to the newly registered band, this email must be sent to the newly registered judge so that they are able to log in to the system once the account has been initialised for them. This is because the administrator must set up the account initially. Included is the judge’s username and their first-time log in password.



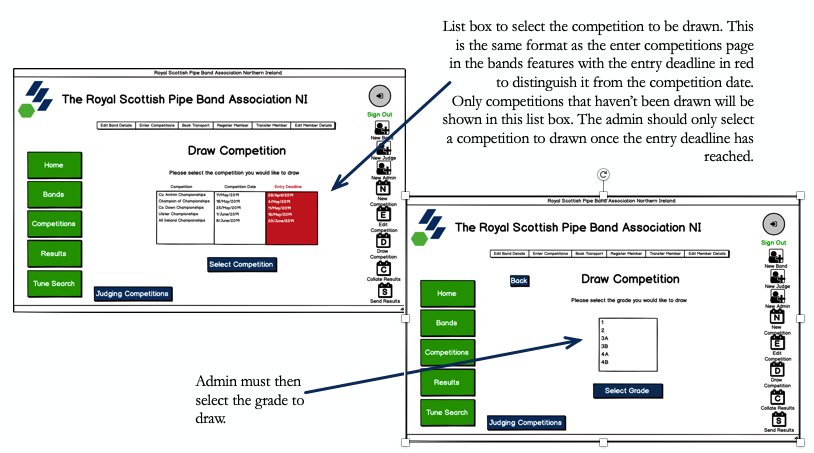
Again, the newly registered admin will also be sent an email with their username and first-time log in password which after an administrator has set up the account initially.

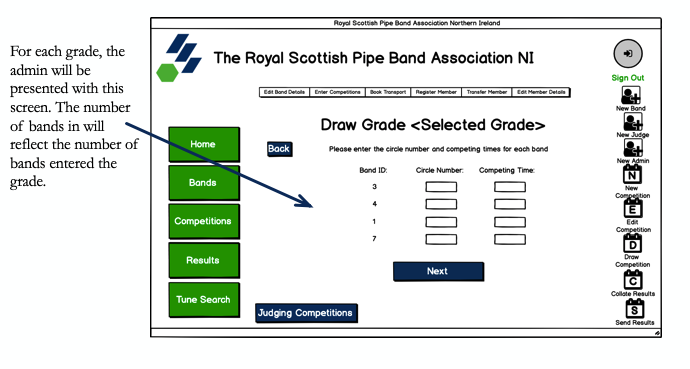


**“Admins have the ability to create, update, delete and draw individual competitions.”**

I decided to design this frame to allow an admin to create a competition, like shown below with drop-down boxes for both dates to stop with any confusion with regards to date format. If I had used an entry box for the date, different administrators could easily use a different date format. The drop-down box only allows the admin to enter data in one specific way, this makes validating these dates much easier.

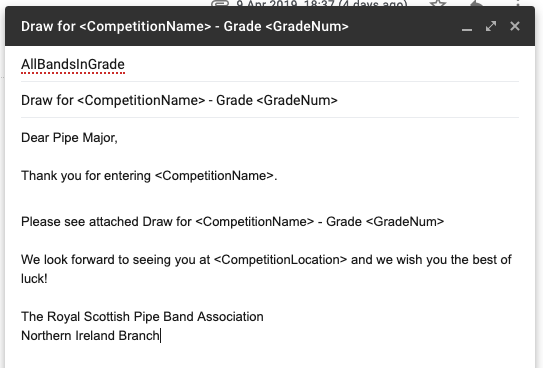




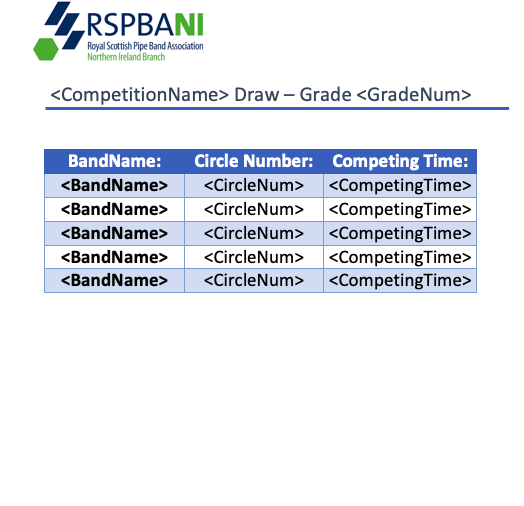


Processing stages

|  |  |
| --- | --- |
| Create Competition | See pseudocode for creating a Band. The code is very similar and instead of saving Band fields to Band Account table Competition Fields are instead saved to the Competitions Table. See validation routines which are used for each field. |
| Editing Competitions | Again, see pseudocode for editing Band details. Competitions page will be populated with the respectful competition fields from the competition table and then once updated the field are checked to ensure they are valid and then old data is overwritten by this new data when saved to Competitions table in database. |
| Drawing Competitions | FUNCTION DrawGrade(CompetitionID, GradeNum, ListOfGrades)  NumBandsEntered = SELECT BandID FROM BandsEntered WHERE CompetitionID = CompetitionID AND Grade = GradeNum  Populate DrawGrade frame. Create an entry box for Circle number and one for competing time for each band in NumBandsEntered.  ON “Draw Grade” Button click  DECLARE ListEmail, Array  FOR I in NumBandsEntered:  INSERT INTO CompetitionDraw (CompetitionID, BandID, CircleNum, CompetingTime) VALUES (CompetitionID, I, CircleNum, CompetingTime)  EntryID = SELECT EntryID FROM BandsEntered WHERE BandID = I AND CompetitionID = CompetitionID.  INSERT INTO Results(EntryID, BandID, CompetitionID) VALUES(EntryID, I, CompetitionID  Email = SELECT Emailaddres FROM BandAccount WHERE BandID = I  ListEmail.append(Email)  Send Email to all bands in the grade to include the draw for all bands.  ListOfGrades.remove(GradeNum)  START  SelectedCompetition = CompetitionSelected from list box  ListOfGrades = “1”, “2”, “3A”, “3B”, “4A” ,”4B”  Until ListOfGrades = “”  Show listbox populated with contents of ListOfGrades  SelectedGrade = Grade selected from next list box presented to user.  DrawGrade (SelectedCompetition, SelectedGrade)  NEXT  END |
| Delete Competitions | START  CompetitionSelected = Selected competition from listbox  On “Delete” Button click  DELETE FROM Competitions WHERE CompetitionID = CompetitionSelected  DELETE FROM CompetitionsDraw WHERE CompetitionID = CompetitionSelected  DELETE FROM BandsEntered WHERE CompetitionID = CompetitionSelected  DELETE FROM Results WHERE CompetitionID = CompetitionSelected  END |

Output:

This is a mock-up of the email to be sent to all the pipe majors in the recently drawn grade. This email will include the name of the competition so that the PM knows exactly which competition they have just received the draw for.

****

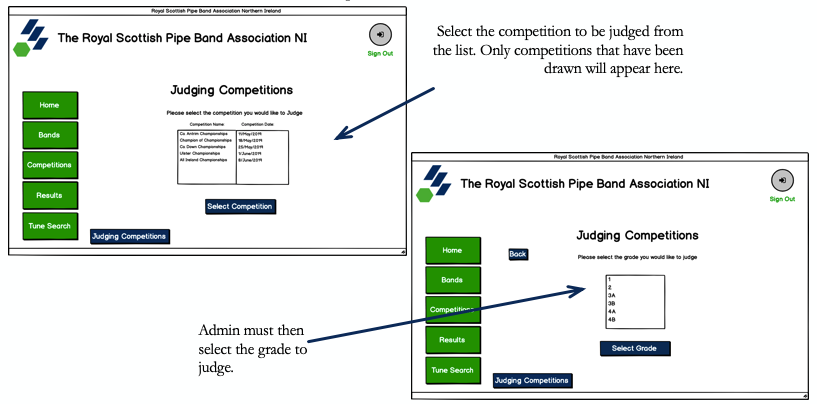
This is the document produced for each grade that has been drawn. It will be attached to every email that is sent to entered bands. The Data seen here is replicated in the Competitions Page in the application.

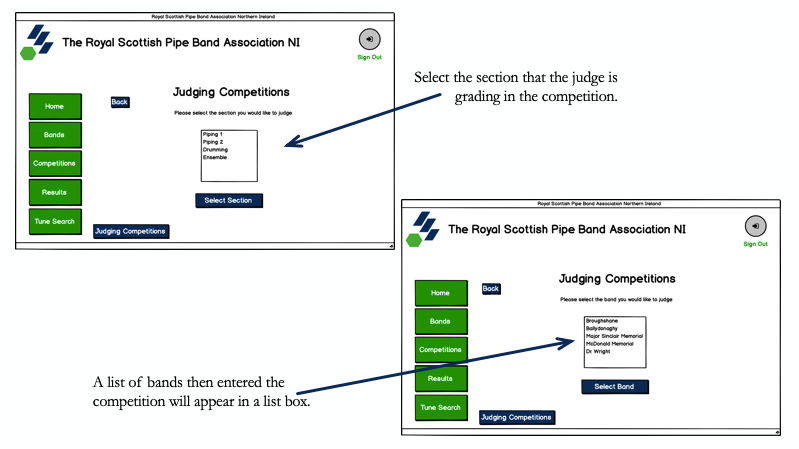
**“Judges have the ability to score bands in competitions.”**

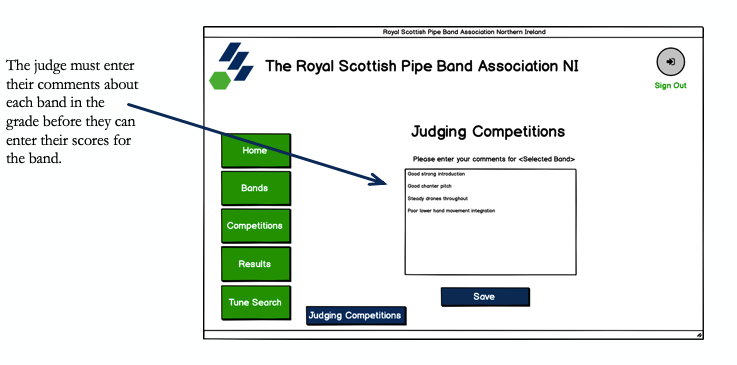
This objective must allow judges to enter their scores and comments about the section of the band that they are grading. It must be possible for the judge to select the competition which they are about to assess. They must easily be able to select the grade and section which they are judging. The program will then take them to a list of bands that have entered that competition from the selected grade. Once the judge clicks save, they will automatically be brought back to the list of bands to be able to select the next band about to compete. Once the grade is finished, they will have the option to go back to the previous pages to be able to select another grade and another section in case they are to assess different sections at the competitions. After all the bands have performed, the grade scores are entered including the best bass section and best march and deportment, they are saved to the database and the total for each band is calculated and the bands are ranked in their positions they have been placed.

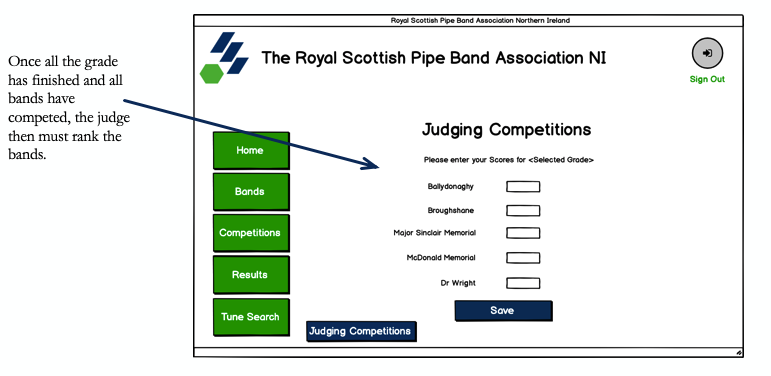
Judge table to get the ID of the judge for reference in the results table.

Once the judge has entered their comments for each section of a band, they must be saved to the database.





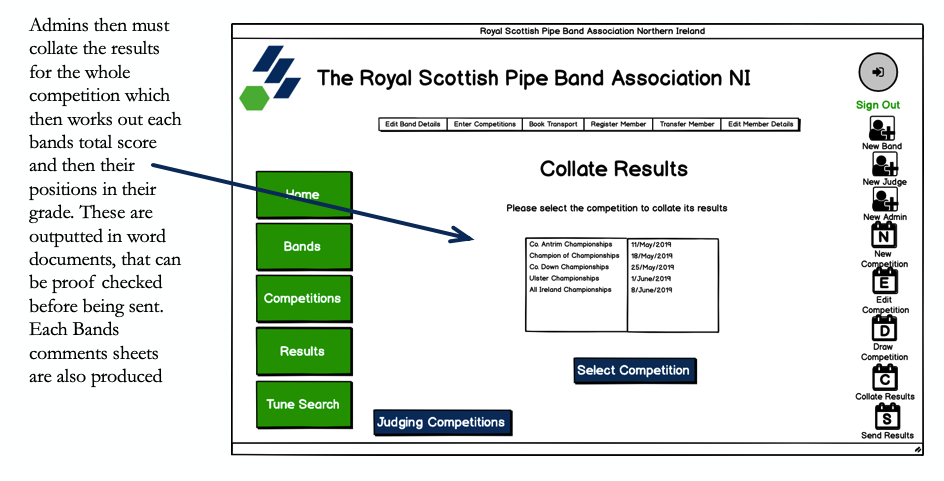
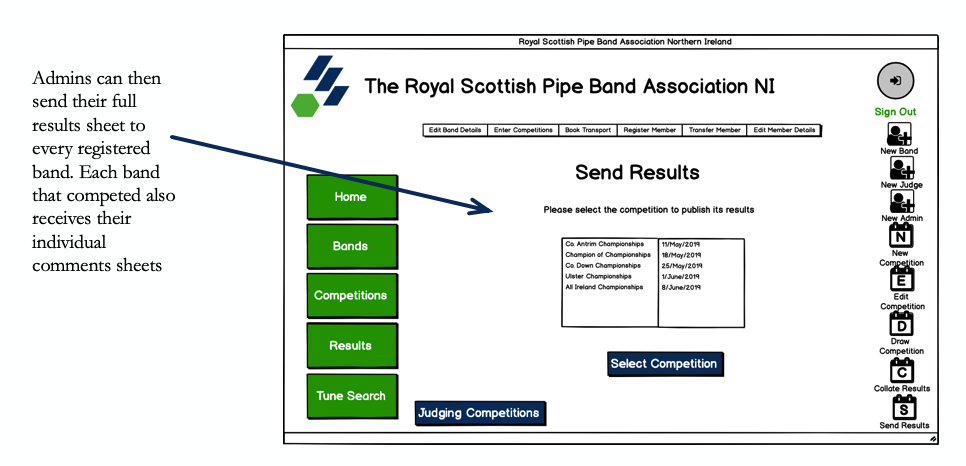




Processing stages

|  |  |
| --- | --- |
| Saving Judge Comments | START  CompetitonID = Selected Competition from listbox  BandID = Selected Band from listbox  Grade = Selected Grade from listbox  Section = Selected Section from listbox  Comments = get comments from textbox  UPDATE Results SET <Section>+Comments = Comments WHERE CompetitonID = CompetitionID AND BandID = BandID  END |
| Entering Score for Grade | START  CompetitonID = Selected Competition from listbox  BandID = Selected Band from listbox  Grade = Selected Grade from listbox  Section = Selected Section from listbox  NumBands = SELECT Count(BandID) FROM Results WHERE CompetitonID = CompetitonID AND Grade = Grade  AllBandsInGrade = SELECT BandID FROM Results WHERE CompetitionID = CompetitonID AND Grade = Grade  FOR I, J in NumBands, AllBandsInGrade:  Display J and Entry box  BandScore[J] = Score  Display Button  On Button Press  Check all scores to ensure they are all between 1 and NumBands and same score is not given to two bands.  FOR I, J in BandScore:  UDATE Results SET <Section>+Score = Score FROM Results WHERE BandID = J AND CompetitionID = CompetitionID  Display list box for judge to select the next grade to compete.  END |

**“Admins then have the ability to collate the judges results and produce the results and send them to all bands.”**

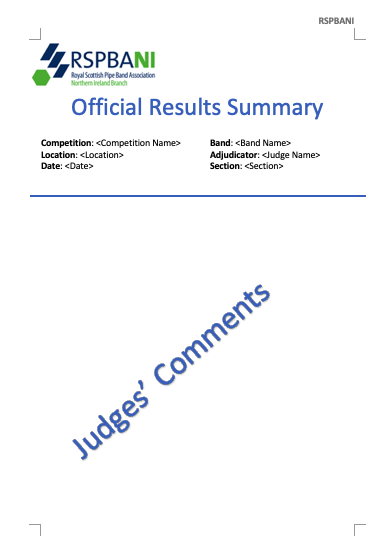
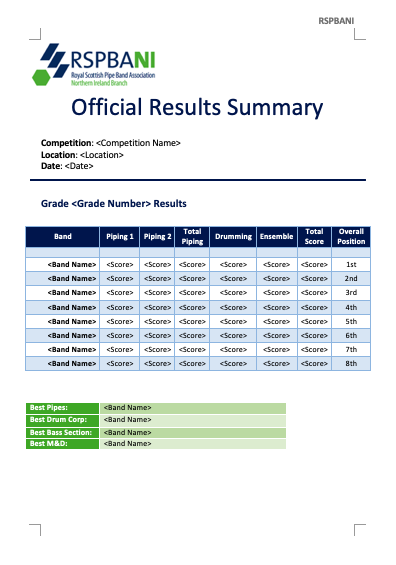


Processing stages

|  |  |
| --- | --- |
| Collating Results | START  # Creation of Comment Sheets  CompetitonID = Competition Selected from list box  ListofBands = SELECT BandID FROM BandAccount WHERE CompetitionID = CompetitonID  For I in ListofBands:  SELECT CompetitionName, CompetitionLocation, CompetitionDate FROM competitions WHERE CompetitionID = CompetitionID  SELECT Piping1Judge, Piping1Comments, Piping2Judge, Piping2Comments, DrummingJudge, DrummingComments, EnsembleJudge, EnsembleComments WHERE BandID = I AND CompetitionID = CompetitionID  SecrtionList = “Piping1”, “Piping2”, “Drummming”, “Ensemble”  FOR section IN SectionList:  Produce new page in document for section. Add Competition Details and section Judge and Comments. (See layout in Judge Comments Sheet output)  SELECT EntryID FROM Results WHERE BandID = I and CompetitionID = CompetitionID  Save document as <EntryID>  UPDATE Results SET ResultsFileName = EntryID WHERE EntryID = EntryID  NEXT  #Creation of full results summary  For Band in ListofBands:  SELECT Piping1Score, Piping2Score, DrummingScore, EnsembleScore FROM Results WHERE CompetitionID = CompetitionID AND BandID = Band  TotalPipingScore = Piping1Score + Piping2Score  TotalScore = Piping1Score + Piping2Score + DrummingScore + EnsembleScore  UPDATE Results SET TotalScore = TotalScore, TotalPipingScore = TotalPipingScore WHERE EntryID = EntryID  GradeList = “1”, “2”, “3A”, “3B”, “4A” ,”4B”  FOR Grade in GradeList  NumInGrade = SELECT Count(BandID) FROM Results WHERE CompetitionID = CompetitionID AND Grade = Grade  RankedBands = SELECT BandID FROM Results WHERE CompetitionID = CompetitionID AND Grade = Grade ORDER BY TotalScore  For Num, Band in NumInGrade, RankedBands:  UPDATE Results SET Position = Num WHERE BandID = Band AND CompetitionID = CompetitonID AND Grade = Grade  CompDetails = SELECT CompetitonName, CompetitonLocation,CompetitonDate FROM Competitions WHERE CompettionID = CompetitionID  Create Full Results Summary Document  ADD CompetitionDetails  For Grade in GradeList  BestPipes = SELECT “G” + Grade + “BestPipes” FROM Competition WHERE CompetitionID = CompetitonID  BestDrumScore = SELECT “G” + Grade + “BestDrum” FROM Competition WHERE CompetitionID = CompetitonID  BestM&D = SELECT “G” + Grade + “BestM&D” FROM Competition WHERE CompetitionID = CompetitonID  BestBass = SELECT “G” + Grade + “BestBass” FROM Competition WHERE CompetitionID = CompetitonID  AllBandResults = SELECT BandName, Piping1Score, Piping2Score, TotalPipingScore, DrummingScore, EnsembleScore, TotalScore, Position WHERE CompetitionID = CompetitionID AND Grade = Grade ORDER BY Position  Create Table on page (see layout in full result summary sheet below)  FOR Band in AllBandResults  Add row to table and populate fields will each field in Band  Create Table, populate will Best Pipes, BestDrum, BestM&D and bestBass  Next page  Save document as <CompetitionID> - <CompetitonName>  UPDATE Results SET CompResultsFileName = <CompetitionID> - <CompetitonName> WHERE CompetitionID = CompetitionID  END |
| Sending Results | #sending Full results summary and Comments Sheets  START  CompetitonID = Competition Selected from list box  ListEmailAddress = SELECT BandID, EmailAddress FROM BandAccount  FOR I, Band in ListEmailAddress:  SELECT CompResultsFileName FROM Competitions WHERE CompetitionID = CompetitionID  Send Email to Band and attach document named <CompResultsFileName >  SELECT ResultsFileName FROM Results WHERE BandID = I AND CompetitionID = CompetitionID  IF results:  Send Email to Band attach document named <ResultsFileName>  ELSE:  Continue  END |

Outputs produced:

Each band will receive a comments sheet for Piping 1, piping2, drumming and ensemble. Each sheet will be populated with the competitions details which will be queried from the competition Table. The adjudicator, section and Judges comments will be queried from the results table and the information stored about the section of the band, in the competition will be used in this output and formatted like below. This data is included so that the band is clearly able to find out which competition and section these comments are about. If I didn’t include these it would be impossible for the band receiving these sheets to work out and understand these comments.

For the official results summary, all the bands in each grade and their results will be shown in the table format below. Again, the competition details are used in this output, so the band knows which competition these results are for. For each grade I will query all the bands and their results, Bands will be ranked in the table by the value of position which will rank the bands from first to last.

Pseudocode for my validation routines

|  |  |  |
| --- | --- | --- |
| Validation Type | Pseudocode | Fields that require the validation |
| Postcode check | Input = TextVar()  PostcodeCheck = False  BEGIN  IF Input.upper() == Format([A-Z][A-Z][0-9] [0-9][0-9][A-Z][A-Z])  OR Format([A-Z][A-Z][0-9][0-9][A-Z][A-Z]) THEN  PostocdeCheck = True  ELSE  OUTPUT(“Invalid Postcode, please re-enter”)  END | JudeAccount.Postcode  AdminAccount.Postcode  BandAccount.BandPostcode  BandMembers.Postcode |
| Presence Check | Input = TextVar()  PresenceCheck = False  BEGIN  IF Input == “” THEN  OUTPUT (“Required Field empty”)  ELSE:  PresenceCheck = True  END | BandAccount.HallName  BandAccount.PracticeTime  BandAccount.Tartan  BandAccount.SecurityQAns  JudgeAccount.SecurityQAns  AdminAccount.SecurityQAns  MembersDetails.Role |
| Alphabetic Check | Input = TextVar()  AlphabeticCheck= False  BEGIN  IF Input.isalpha() == True THEN  AlphabeticCheck = True    ELSE:  OUTPUT (“Field requires alphabetic letters only”)  END | BandAccount.AddressLine2  AdminAccount.AddressLine2  JudgeAccount.AddressLine2  BandAccount.FirstName  AdminAccount. FirstName  JudgeAccount. FirstName  BandAccount.SecondName  AdminAccount. SecondName  JudgeAccount. SecondName |
| Password Check | Input = TextVar()  LengthCheck = False  LowerCheck = False  UpperCheck = False  PasswordCheck= False  BEGIN  IF length(Input) <= 8 THEN  LengthCheck = True  For letter in Input  IF letter.islower == True  LowerCheck = True  IF letter.isupper == True  UpperCheck = True  IF LowerCheck == True  AND UpperCheck == True  AND LengthCheck == True THEN  PasswordCheck = True  ELSE  OUTPUT(“Password invalid”)  END | BandAccount.Password  AdminAccount. Password  JudgeAccount. Password |
| Username Check | Input = TextVar()  PresenceCheck = False  ListOfUsernames = []  UsernameMatch = False  UserNameCheck = False  IF PresenceCheck(Input) == True  PresenceCheck = True  AdminUsername = SELECT Username FROM AdminAccount  For Username in AdminUsername  ListOfUsernames.append(Username)  JudgeUsername = SELECT Username FROM AdminAccount  For Username in JudgeUsername  ListOfUsernames.append(Username)  BandUsername = SELECT Username FROM AdminAccount  For Username in BandUsername  ListOfUsernames.append(Username)  For user in ListOfUsernames  IF Input == user  UsernameMatch = True  IF UsernameMatch = True  AND PresenceCheck = True  THEN  UsernameCheck = True  ELSE  OUTPUT(“Username is already in use”) | BandAccount.Username  AdminAccount. Username  JudgeAccount. Username |
| Email Address Check | Input = TextVar()  EmailAddressCheck = False  AtCheck = False  DotCheck = False  FOR letter IN Input  IF letter ==”@” THEN  AtCheck = True  IF letter == “.” THEN  DotCheck = True  IF AtCheck == True  AND DotCheck = True THEN  EmailAddressCheck = True  RETURN True  ELSE  OUTPUT(“Email address is not valid”) | BandAccount.EmailAddress  AdminAccount.EmailAddress  JudgeAccount.EmailAddress  MembersDetails.EmailAddress |