# Lost Ark Template Saver

By

Jonathan Chan

January 8, 2022

## Abstract

Within Lost Ark Online, either due to the phenomenon of fear of missing out or actual enjoyment of playing the game a player may have many characters of different classes and builds, the problem may arise where a player does not want to waste precious brain storage to memorize which skills apply which status effects upon the enemy and would like a tool to configure and reference to.

The solution was to develop an interface to allow the user to initialize and customize templates with simply colours and text for quick reference. The templates themselves are a static layout for uniformity and, honestly, lessen the complexity of the system. For organizational purposes, the user is able to group the templates into template groups. The encompassing system includes a graphical user interface application with an internal local database to store the templates' information.

## Acknowledgements

I would like to acknowledge and thank all my friends and acquaintances that played Lost Ark with me for any length of time. They bettered my experience and provided a nice respite to my crushing impulsion to “keep up” and be on the bleeding edge. As they understandably left this void wasteland of a game, I continued gripped by the addiction of progression; of getting more “qualified.”

I do have to acknowledge the game that is does provide some very enjoyable moments and many of the systems have acceptable design, but it impressively does a fantastic job at somehow eliciting the same conclusion from a large number of the players; “this game is a job.” For a while, it was fun for my miserable brain to latch onto to and in the end I eventually I held the same sentiment.

## Contents

## List of Figures

# Chapter 1 - Introduction

During ones journey playing the MMO Lost Ark Online, they may find themselves, possibly with no volition of their own, having a multitude of characters that have to perform activities related to combat and therefore may not always instantly recall what each skill's effects are. Having a simple application that is configurable to display each hot key's combat ability's effects may be beneficial to a user that needs a quick review of the character's current layout or can glance at it when they need an ability with a specific effect.

## 1.1. Problem Motivation

Of course, a valid question is, why not just hover the cursor over the abilities over each hot key slot and it'll provide all the information of that ability? Well yes, that is an effective, logical, and built in approach to refreshing the user on a combat ability's information, but sometimes they do not want all the details and it feels time consuming. Ultimately, they want to quickly know the effects important to them. Having an application that allows the user to freely configure and customize, within the bounds of the application, hot key templates that portrays what effects each hot key has can benefit the user and expedite one of the key steps in the process of preparing for combat or even during combat. An example is the effect "weak point," if a raid's mechanic requires a certain number of weak points to be inflicted on the enemy in a certain amount of time; it would be imperative for the player to know which skills have it, the levels, and with that what order to cast. A sample order would just be highest to lowest level of

weak point, in case the time restraint is strict enough that the player cannot use all the qualifying skills.

## 1.2 Problem Statement

This project solves the issue of manually reviewing each skill's effects the user may have forgotten. This skill preview application allows the user to create and customize static templates of the hot key locations of combat abilities with basic text and colour coding that indicates information most important to them. Each configured template should be able to be retrieved and accurately displayed to the user for speedy review or modification.

## 1.3 Proposed Solution

The final product graphical user interface will be a Windows desktop application (I want to try GUI development with C# NET) and should provide the user ease of access when creating template groups and then creating and customizing individual templates within that group. Each template should statically display 10 buttons to represent the hot keys primarily used in combat and a section for general notes the user may want to record for the template.

The templates' information will be stored locally in the directory of the application (SQLite). This is to remove the requirement of a web server to host a database for the application.

## 1.4 Accomplishments

The accomplishments toward the solution were the implementation of a graphical user interface and a local database to create a system that allows the user to create, customize, and organize template groups and templates for the users' needs.

## 1.5 Organization of the Report

This report includes the methods and descriptions of how the solution was designed and then implemented to create a fairly basic application for Lost Ark skill template storage. The technical solution section of this report presents brief explanations of the design processes that contributed to the overall solution.

# Chapter 2 – The Engineering Project

## 2.1 Health and Safety

This system may ease the burden of memorizing the skill configurations for the player's multitude of Lost Ark characters. The negative side effect is that this could lead the player to believe that actively dedicating time to more than one character is now more manageable, but the unpleasant truth is the game rewards the unhealthy addiction of playing more characters to capitalize on the phenomenon of fear of missing out. On the other hand, a healthy player that

doesn’t obsessively play may find this tool useful when they return to the game from time to time.

## 2.2 Engineering Professionalism

This project had no hard deadlines due to being a personal project, but in hindsight there should have been to enforce some discipline in order for the project to be completed more promptly. Even though I wrote the specifications they were developed from a client’s perspective to ensure the system would satisfy other users, not just a system that required the bare minimum functionality and haphazardly thrown together. The one unprofessional practice was, during development, if I realized a feature I had initially intended to include was too complex I just wrote it off as out of scope and removed it. The major ones were more hotkey button customization and version template data portability, if fundamental database structure changes were different between GUI applications.

## 2.3 Project Management

The project’s management loosely used the software process model of Phased – Release. The requirements and specifications were decided early on and then through phases each operational deliverable was implemented and then tested to satisfy user acceptance.

## Chapter 3 – Background Literature Review

As reiterated in previous sections, players may need a reference to which combat hot keys impose what status effect on an enemy. The default combat hot keys are as follows: Q, W, E, R, A, S, D, F, Z, X, and, V. Two effects that I personally would like to have a reference for are “Weak point” and “Stagger”, these also have levels of their potency. Weak point has level 1, 2, and 3, whereas stagger has low, mid, high, and highest. As you can imagine, if a certain effect is required to be applied to the enemy then the correct hot key must be pressed.

## Chapter 4 – Technical Solution

A brief summary of the project’s implemented solution, on start up the application will verify if all the database tables in the system are present and accounted for as a very basic level of verification. I have determined that automatic correction of malformed data is outside the project’s scope due to either the error was incurred from a function’s incorrectness or direct manipulation of the date in the internal database. If an error occurs and the user (me) doesn’t want to do a factory reset it can be manually investigated and if the error stemmed from a function, it will be corrected. On failure, it will display to the user 3 options; “Close”, “Template export”, and “Factory Reset”. “Close” will exit the application, “Template export” will export all information from the tables that do exist to a text file in the project’s file directory, and “Factory Reset” will drop all tables and create them anew. On verification success, the application will initialize the main window for the template saver application. At this point the user has the ability to create or select template groups and then create or modify individual

templates within the selected group. A few screenshots of the implemented solution can be seen in Appendix J.

The project's solution required a system with two components to provide the user with a local and offline application to enable them to customize and save templates. Firstly, the graphical user interface was essential to provide a visual representation of the templates and interact with them. The second was the local and offline database that would contain all the template information for the application to save to and source from. The reasons that influenced the decision to produce a local and offline solution is that I wanted to create a project in C# NET in visual studio and did not want to host an application and server for the database. A drawback of this offline solution is that, since users have to download the executable and the database is internal to the application, there could be multiple versions of the system floating around so there would be a need for functionality to port the database information to different versions, forward and backward, in the case that the database structure is changed. This was left on the cutting room floor due to I will probably be the only user and I wanted to complete the project before I became remarkably demotivated and drop it. Compared to a server based solution, since the system would be centralized, the upgraded versions would apply to all users.

The design phase included many of the components fundamental to software design and were created from the perspective of a client through requirement elicitation. It is paramount to have all design elements agreed upon and concisely documented before the time of

implementation, but if changes are required during implementation then ample time must be provided to apply them before internal testing and client validation testing.

The initial step was to design the graphical user interface, the interface mockups were created to present a visual representation and starting point for the GUI to implement, and these mockups can be seen in Appendix A. The major design considerations were to keep the interface as simple as possible so that it was intuitive to use. A design idea implemented to accomplish this was to ensure the main application window that contained the template groups and templates displayed every component stacked from the top to bottom to represent some sense of order. For example, within the top header section the template group can be chosen and then all associated templates are loaded below in the designated template section. The individual template's button layouts are statically designed to represent the physical placement of the default keyboard key layout of the combat skills; Q, W, E, R, A, S, D, F, Z, X, and, V. Since the button layout is static and the user could change the key binds in game, the buttons in the template layout have a section to allow the user to label the button. The additional customizations to the buttons are simple, but effective for referencing purposes, such as choosing one colour for the button and text within the button itself. Under the hot key layout, there is section for general notes and they could save, for example, the order for stagger skills; "S > D > W > E." The name of the template and the order of the template within the group can be modified. An important feature each template has is to transfer the template to another group; it will just append it to the end of the destination group. The primary reasons to have a

static layout for each template is to enforce uniformity and to reduce the complexity of the system in terms of storing additional amount information per template to display more configurable template layouts, like element size, position, and etc.

Regarding the implementation of the database, this project's database was decided to be a local and offline database so that an external server did not have to host the database. The database engine employed was SQLite and its database management system is relational. The design of the database tables leverage the feature of relationships to logically connect a template group to its templates to each template's attributes, the database structure can be seen in Appendix B.

The specifics of the functional requirements can be seen in Appendix C, non-functional requirements in Appendix D, and scope of the system in Appendix E.

The scenarios of how the application may be used can be seen in Appendix F, the use case diagram of what functions within the system the user may use can be seen in Appendix G, the actors of the system are listed in Appendix H, and finally the specific use cases that the user may perform through the interface are described in Appendix I. The before fore mentioned design steps combined are a necessity to the software's design and implementation, it describes in logically ordered steps on how the user would interact with the application to

invoke certain functionalities and how the system responded to eventually reach the desired outcome. For example, adding a template to a template group and then customizing a template. For the black box testing, the measurement of success was it performed the scenarios with the desired results without negative side effects. On the other hand in a weak form of white box testing, the use cases were performed to validate the interaction between the actors and the system. Another testing methodology that was not properly employed was unit testing. Both white box and unit testing were unofficially tested during implementation to eventually meet the requirements described in the use cases.

## Chapter 5 – Conclusion

The problem may arise that the player needs a reference to which skills do what and at what effectiveness, notably weak point and stagger. During combat the game may require the player to apply these statuses on the enemy to invoke a reaction or clear a set mechanic.

The solution was to create a system that included a graphical user interface and a database to allow the user to add, configure, and organize statically generated templates so that they can be used as a reference.

The accomplishments were that I implemented the solution fulfilling the requirements I set out to satisfy. The graphical user interface was implemented using C# NET in visual studio and serves as an acceptable tool for referencing. It allows the user to create template groups for organization and add then customize templates with simple options like the buttons with

colour and text. The local database used for the system to store the template information is SQLite. I do not think that this solution has any features that make this any more desirable than other existing solutions that address this problem.

If I were to expand on this solution and kept it offline, I would consider adding the features I cut out. I would consider adding the portability feature, in case the database structure did change between different versions of the system, it would probably just be hard coding if going to X version from Y, then migrate the database information in a specific way. In terms of customization to give the user more freedom, I would want to be able to at least move around the hot keys, add more buttons, and allow the button to be set to more than one colour.

To reiterate, the reason I chose an offline system is that I wanted to experience implementing a project with C# NET with visual studio because I haven't yet built one. It also helps that, while doable, I don't have to host the web application, the web code and database. I do believe that a web application is the more appropriate solution for this problem; more convenient for the user, better application maintainability, and etc. The user would just go to a web link and log on for their templates and not needed to download the program, say from Github. The current state of the offline system has no maintainability due it would have to be individually downloaded and porting template data version to version is not slated as a feature. The advantage of a web application is that all users would view the same website and if there were any database structure changes on the new version, the developer would need to ensure the

data was ported and it was compatible with the website. The presentation of the template saver would definitely look more visually appealing due to vast HTML and CSS libraries.

I will not make excuses, my skills have dulled or they were never there in the first place and it may be evident in the design and implementation. I tried my best to enforce best practices like code reuse and properly scoped variables, but I am sure I missed the mark. Onward I implemented each deliverable, it works, but it’s not a perfect solution.

I don’t foresee returning to Lost Ark at the capacity I previously allowed myself to commit to it and I am content with the accomplishments of this project. These two roads have converged and I move forward.

This project is concluded.

# Appendices

## APPENDIX A - Graphical User Interface Mockups

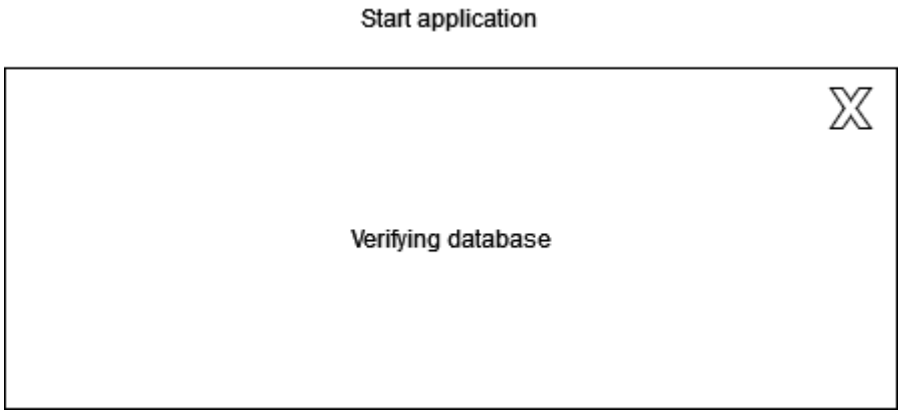


*Figure 1: Start Application Window*

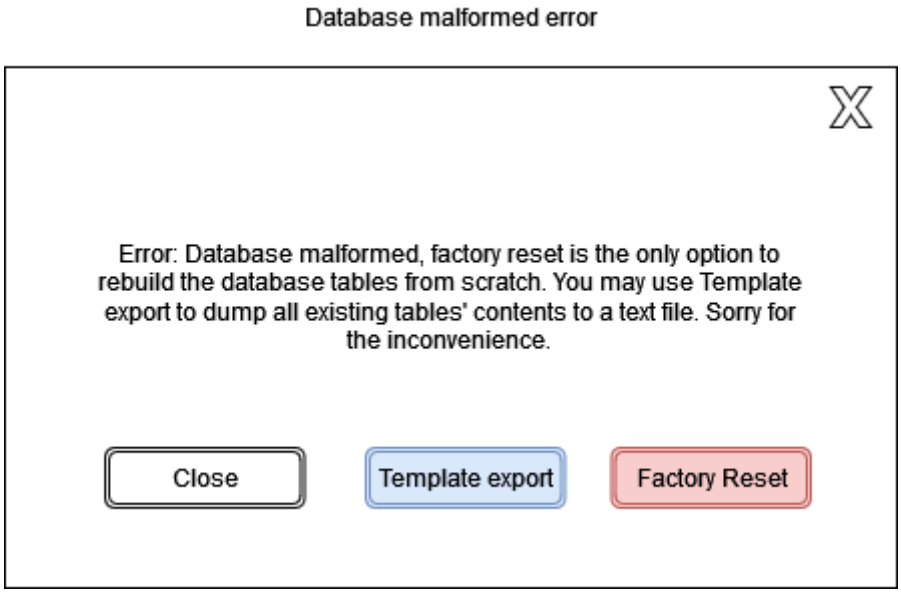


*Figure 2: Database Malformed Error Window*

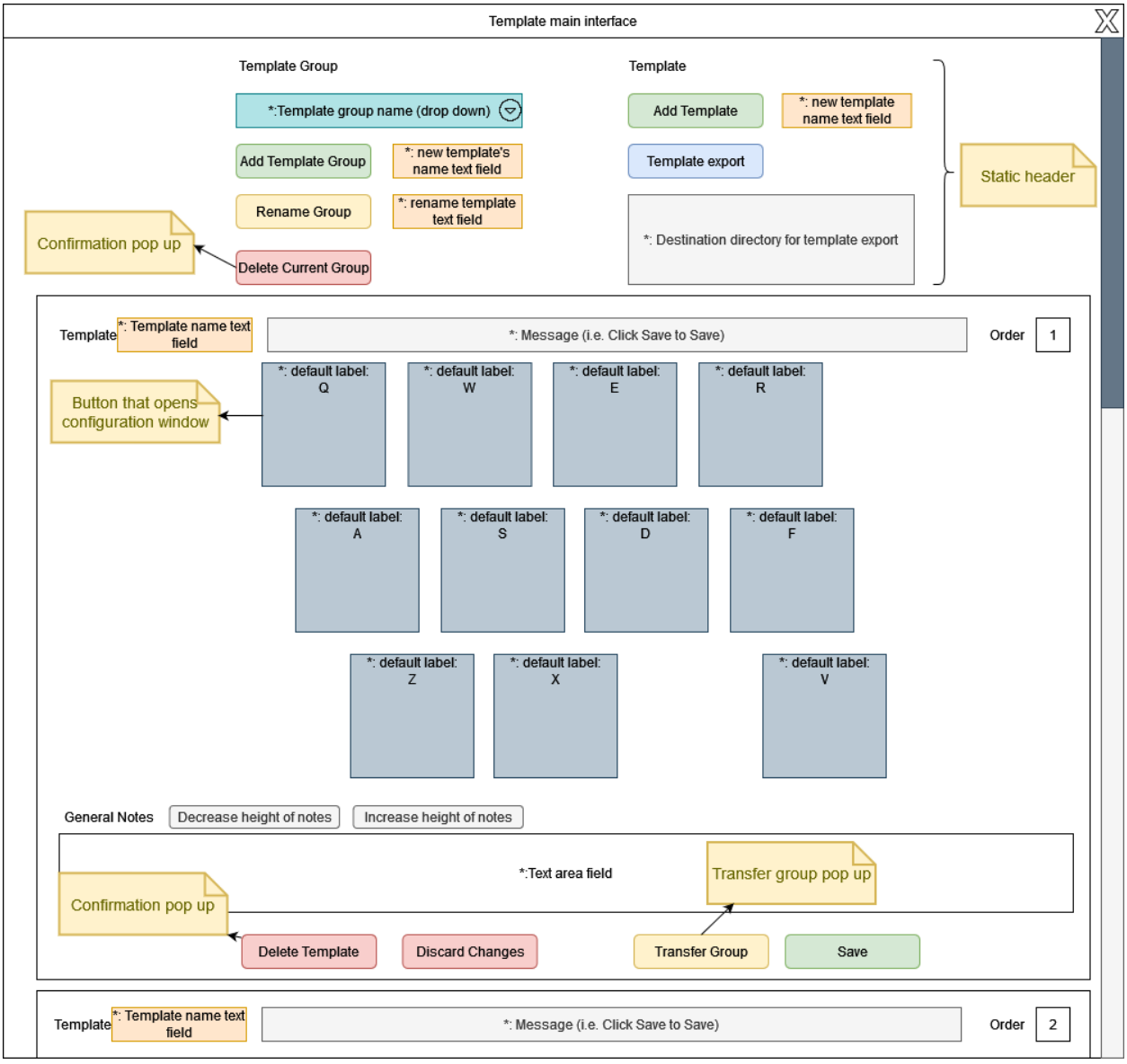


*Figure 3: Main Template Window*

Template Group deletion confirmation pop up window

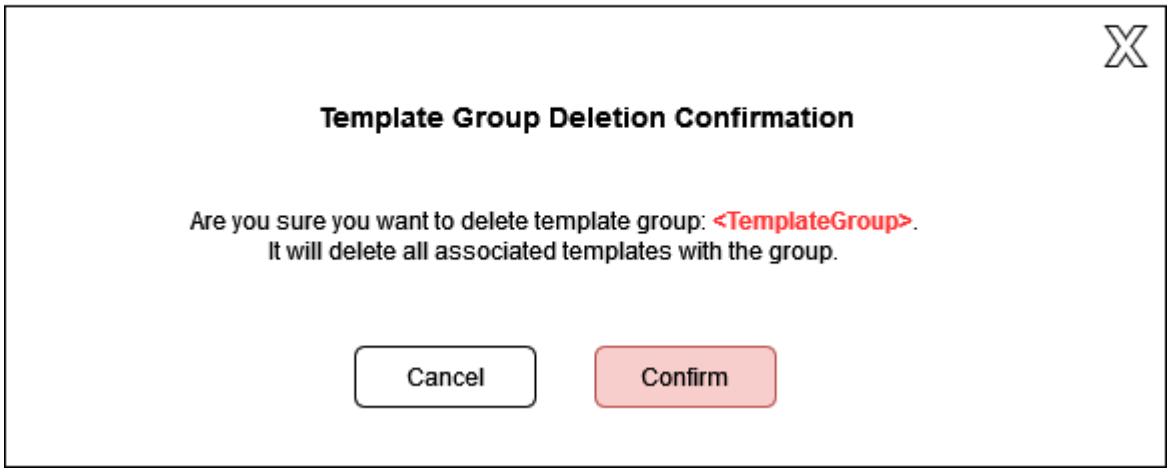


*Figure 4: Template Group Deletion Confirmation Window*

Template deletion confirmation pop up window

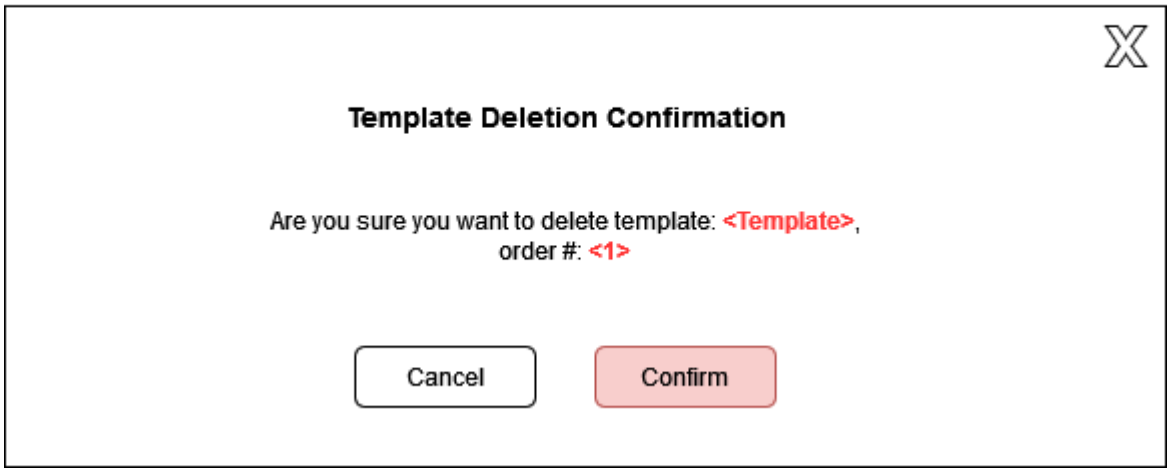


*Figure 5: Template Deletion Confirmation Window*

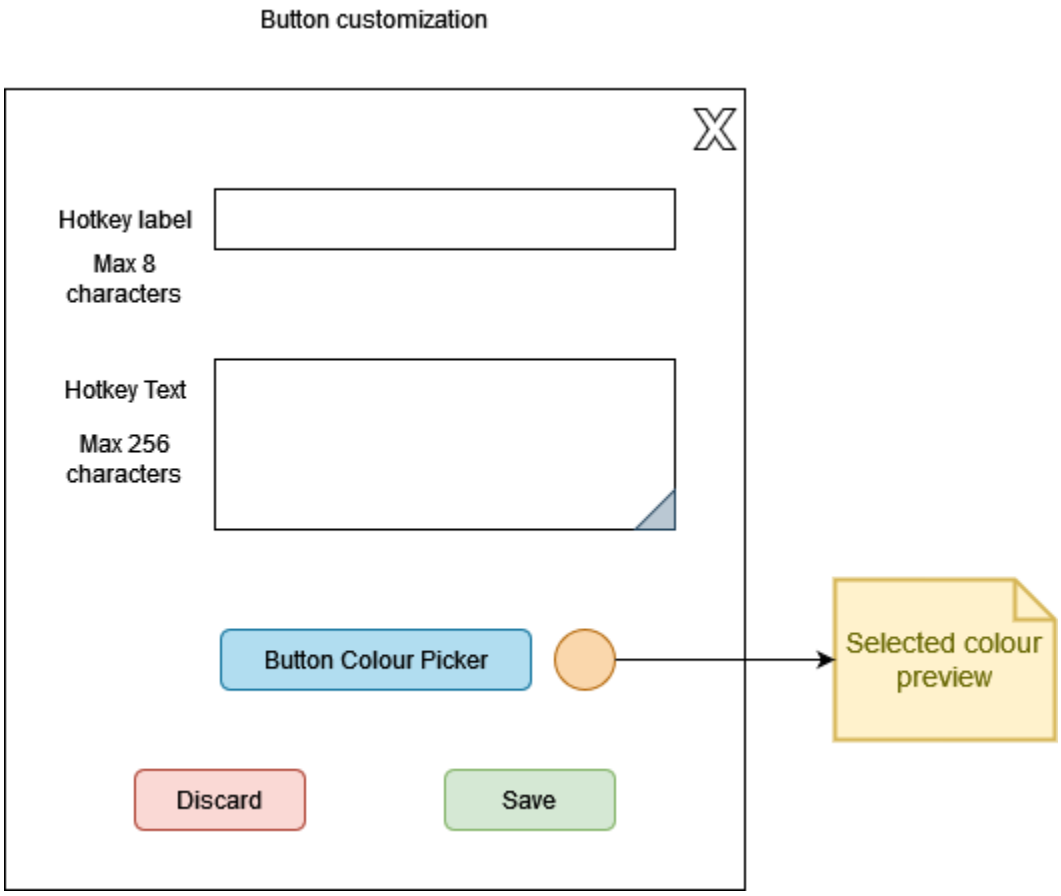


*Figure 6: Button Customization Window*


*Figure 7: General Error Message Window*

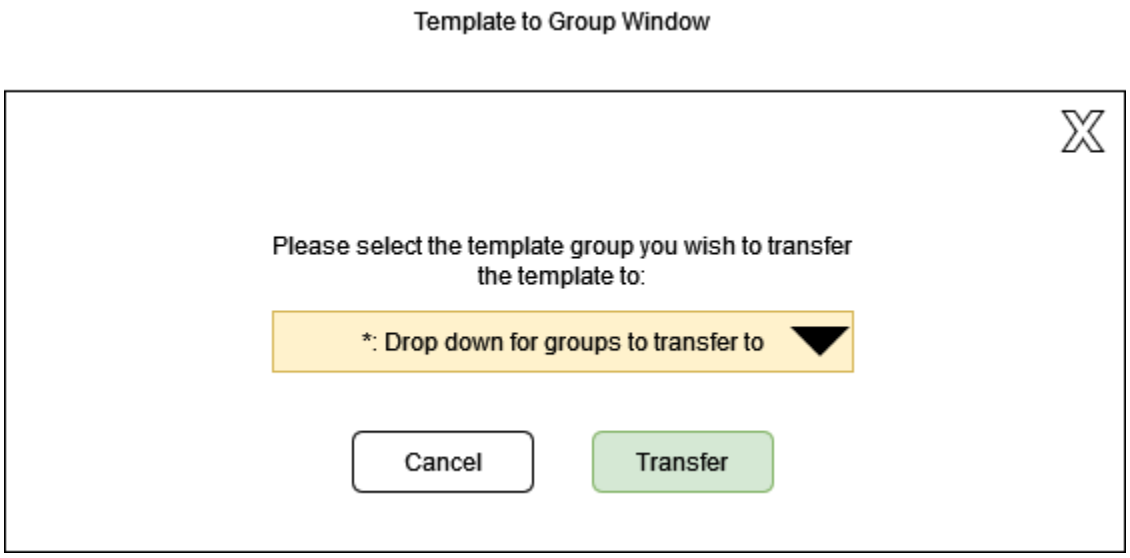


*Figure 8: Template to Group Window*

## APPENDIX B – Database Structure

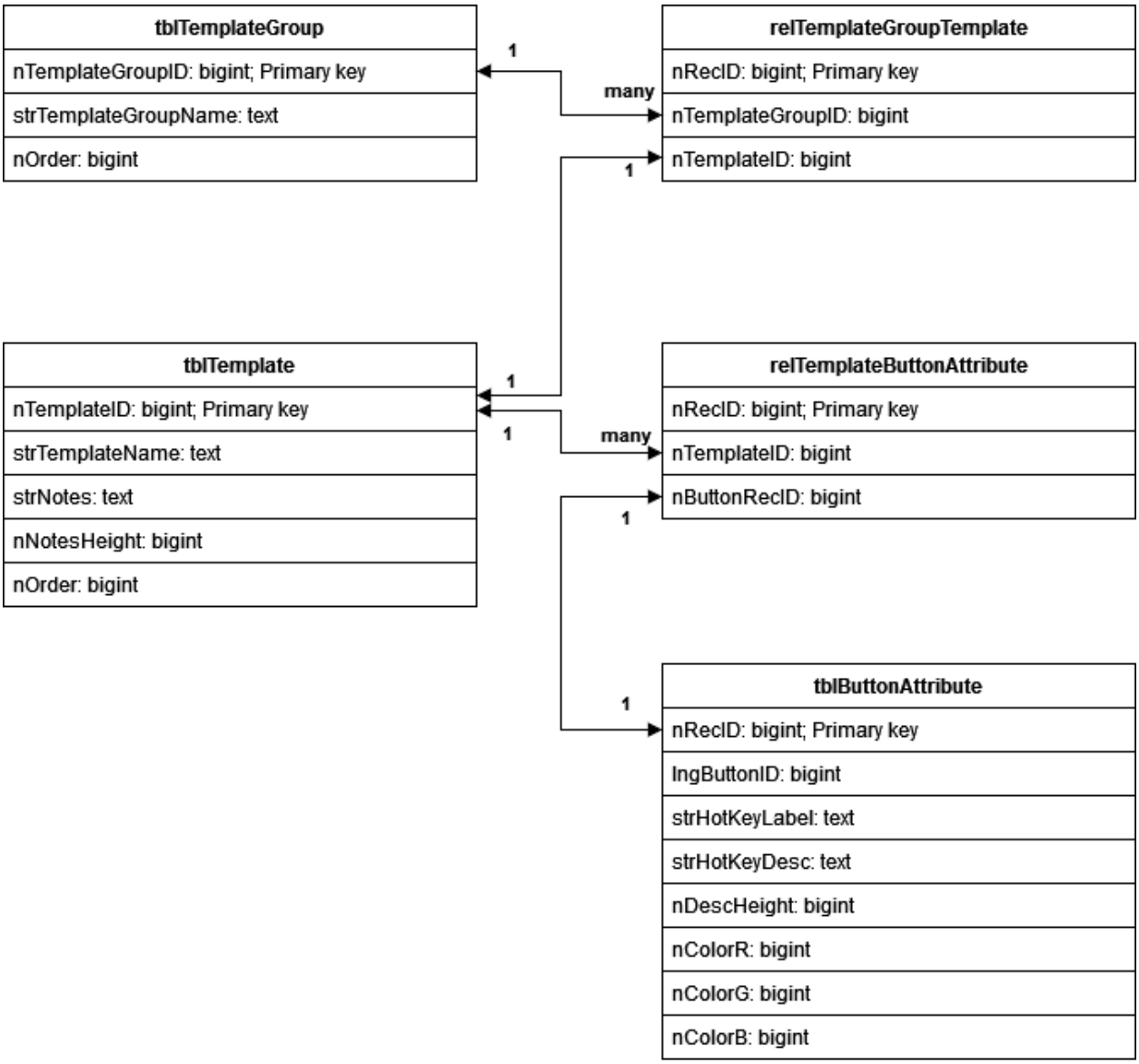


*Figure 9: System's database structure*

## APPENDIX C - Functional Requirements

- Actors:
    - User: Add, delete, update, and view templates through the GUI

    - Template database: Database to store all information relating to the templates to populate the GUI
- The application should provide a graphical user interface, GUI, to the user to access the templates.
- The application should allow the user to create a template group
- The application should allow the user to add a template to the template group
- For each template, the application should allow the user to customize the buttons in the statically generated layout of buttons.
- The button customizations should include a text field of the button label, main notes for the button, and the colour of the button.
- The colour of the button should have the feature to choose from a colour picker.
- For each template, it should include a input field to allow the user to set the order of the template in the list.
- For each template, it should include a section for the user to add general notes.
- For each template, it should include the options to “Delete Template”, “Discard Changes”, “Transfer To Group”, and “save”.
- For operation “Delete Template,” that entire template should be deleted and removed from the template group
- For operation “Discard Changes,” no information pertaining to that template should be updated in the database. The original settings will be reloaded.
- For operation “Transfer To Group,” the template should be assigned to the selected template group destination and at the bottom of its template list.

- For operation “Save,” all information pertaining to that template should be saved in the database.

## APPENDIX D - Non – functional requirements

GUI:

1. Usability:
    a. The main application window’s components and containers, that presents the templates, should be sized to be easily viewed on half the width of and 1920x1080 p monitor, which is 960 p wide.
    b. There should be built in help section to contain documentation on how to operate the interface.
    c. There should not be a maximum number of windows allowed to be active, due to the user may want to edit many buttons in parallel.
    d. The buttons for each template should not be manually resizable.
    e. The text area for a template should be manually resizable in terms of height and it should be saved for future viewing.
    f. When the main template window is resized below the minimum height or width of the container of the then the appropriate directional scroll bars should appear for use.
2. Dependability:
    a. Reliability:

i. The application should always open successfully, even in the event that the database has an issue. In the event of an issue, the application will present the user with the option to export the current information to a text file and factory reset the tables.

ii. The application should always display the correct template groups in the template group drop down menu and display the associated templates in the templates section.

iii. The application should always accurately save the user's template customizations.

iv. The application should always accurately display templates' customizations, which include the name, order, button customizations, and general notes, and general notes height.

b. Robustness:

i. Restrict the length of characters in specific input fields:

1. Hotkey label, restricted to 8 characters
2. Hotkey Text, restricted to 70 characters
3. Template group name, 64 characters
4. Template name, 64 characters
5. General Notes, 1024 characters

ii. Restrict data type in specific input fields:

1. Template order field, integer only.

c. Safety:

i. The application will prevent SQL injection in all text fields.

1. Hotkey label
2. Hotkey internal text
3. Template group name
4. Layout name
5. General Notes

3. Performance
    a. Response time:
        i. The response time to save or display a particular template will be no more than 1 second
    b. Availability:
        i. The system and all its components will always be available to service the saving and displaying of templates
    c. Accuracy:
        i. There will be 0 magnitude of error in saving or displaying of information pertaining to the templates.
4. Supportability:
    a. Adaptability:
        i. N/A. Porting template data to different versions out of scope.
    b. Maintainability:

i. This product will be readily portable to the Windows 10 operating system.

ii. Since the system is an offline desktop application, as mentioned in Adaptability, to receive the most recent version the user must manually download the executable.

## APPENDIX E - Scope of the System

1. Offline Lost Ark template system: Local desktop application that allows the saving and displaying of templates that can be customized by the user. When a template is modified and chosen to be saved, it is saved in a database. When displaying the available templates from the selected template group, it will retrieve from a database.
2. One database is envisioned: Database to store the data of a template in a manner that the interface can interpret to display when required.
    a. Scope of the database:
        i. Internal database: The "user entity" uses the interface that has direct functionality to modify data in the database. Therefore, we must design the SQL table structures and API for all functions. Not for "outside use" and it holds data that is created/updated/accessed by this application.
        ii. The database solution is SQLite and is a component that will be configured and deployed locally for this system but not designed.
3. Summary of the scope:
    a. IN the system to design:

i. Graphical user interface for the management of templates

ii. Database to store the templates' information

iii. Application programming interface for the GUI to interface with the database.

b. OUT of the system

i. User entities are the actors that trigger the behavior of the system

c. Responsibility of the system:

i. Reliably and correctly display the templates that are available to the user on the interface

ii. Correctly saves the user's templates, either new or modified.

d. What the engineer has to build:

i. Design the interface layout for ease of access

ii. Design the database table structure in a modular manner so that potential changes to the structure can be easily applied.

iii. Design the API so that template information is stored and retrieved in a concise and organized manner.

## APPENDIX F - System Models - Scenarios

**F.01 User desires to create a template and configure it**

Actors:

- Bob

Scenario:

- Bob determines that is would be convenient to have a template for their character's skills to quickly remember which hot keys have what effect.

Source of information:

- Through observation and personal experience, the user would use the system to configure templates in such a way to convey information important to them.

Details:

- While playing Lost Ark, Bob notices, for one reason or another, that they forget what skills have the effect 'stagger' and what levels of effectiveness they are.
- Bob opens the template application and selects the desired template group. Bob adds a new template, they customize the hot keys with stagger with the colour yellow and sets the hot key's inner text with a number to indicate stagger level, i.e. For low they entered 1 and highest they entered 4.
- In the general notes section under the hot keys, Bob also enters a note about the rotation of skills order for the highest stagger to lowest so that it is easy to refer to.
- Bob clicks 'save' at the bottom of the template and it ready for future use.

**F.02 User desires to delete a template**

Actors:

- Jonny

Scenario:

- Jonny determines that a template that was previously saved is out of date and has no future use.

Source of information:

- Through observation and personal experience, a user may not want a template to exist anymore. It could be any amount of change in the in-game skills hot key configuration that causes the template to be obsolete.

Details:

- Jonny determines that they desire to change his character's configuration of skills and completely overhauls its skills and has no intention of returning to the old configuration.
- Jonny selects the template group the template slated for deletion belongs to.
- At the bottom of the template Jonny selects 'Delete', a confirmation pop up window appears and Jonny confirms.
- The template is completely deleted from the system.

**F.03 User desires to group templates for better organization**

Actors:

- Dmitry

Scenario:

- Dmitry has templates that are associated with the same character and they desire to group them together to have them organized.

Source of information:

- Through observation and personal experience, a user may have multiple configurations for a character, i.e. Raids, Chaos, and etc., so that they create multiple templates to house the configurations. Allowing the user to group templates into template groups allows some form of organization.

Details:

- Dmitry has multiple templates created for one character and finds it distressing to have other non-relevant templates together.
- At the top of the interface, Dmitry creates a new template group with the name of his character.
- In the default general template group, he finds each relevant template and at the bottom he clicks the ‘Transfer group’ button. A pop up appears with a selection drop down to prompt the user which template group they would like to move the template to, they select their newly created template group and confirms.
- After Dmitry has transferred all their templates to the new template group, they do not appear within the default general group. Dmitry selects the new group and the application displays the correct templates.

## APPENDIX G - System Models - Use case diagram


*Figure 10: Use Case Diagram for Template Saver*

## APPENDIX H - System Models - List of Actors

1. User: Has access to all functionality of the template's system interface.
2. Template database: Stores information pertaining to the templates configured by the user.

## APPENDIX I - System Models - Use cases

### I.01 Start application

1. Brief description: User starts the application.
2. Primary Actor:
    1. User
3. Precondition:
    1. None
4. Secondary Actor:
    1. Template database
5. Dependency:
    1. EXTENDED BY USE CASE Database malformed
    2. EXTENDED BY USE CASE Factory reset
6. Basic flow:
    1. User executes the application executable.

2. The system verifies that the `template database` tables are all present and accounted for.
3. The system initiates the template configuration interface.

7. Post condition:
    1. `User` is presented the interface that allows for template configuration.
8. Specific Alternative Flows:
    1. RFS Basic flow 2:
        1. IF the system finds that there are any tables missing from `template database`, THEN ABORT, ELSEIF RESUME STEP, ENDIF
        2. Post condition: The template export and factory reset options are presented to the `user`

**I.02 Add a template group**

1. Brief description: Initiate the creation of a new template for the user.
2. Primary Actor:
    1. `User`
3. Precondition:
    1. Use case ‘Start application’ successful
4. Secondary Actor:
    1. `Template database`
5. Dependency:
    1. None

6. Basic flow:
    1. `User` enters the desired template group name in the text box beside the button ‘Add Template Group’.
    2. `User` clicks the button ‘Add Template Group’.
    3. The system creates the template group in the `Template database`.
    4. The system selects the new template group as the active template group.
7. Post condition:
    1. The created template group is displayed and included in the select drop down menu for template groups.
8. Specific Alternative Flows:
    1. None

**I.03 Modify a template group’s name**

1. Brief description: The template’s group name can be modified after creation.
2. Primary Actor:
    1. `User`
3. Precondition:
    1. Use case ‘Start application’ successful
    2. Use case ‘Add a template group’ successful
4. Secondary Actor:
    1. `Template database`

5. Dependency:
    1. None
6. Basic flow:
    1. `User` selects the template group from the select drop down menu.
    2. `User` enters the desired new name into the text box beside the button labeled ‘Rename Group’.
    3. `User` clicks the button ‘Rename group’.
    4. The system updates the template group’s name in the `template database`.
7. Post condition:
    1. The template group’s name is updated and presents the updated name.
8. Specific Alternative Flows:
    1. None

**I.04 Delete a template group**

1. Brief description: An entire template group is to be deleted; this includes all templates that associated to it.
2. Primary Actor:
    1. `User`
3. Precondition:
    1. Use case ‘Start application’ successful
    2. Use case ‘Add a template group’ successful
4. Secondary Actor:

1. Template database

5. Dependency:
    1. None
6. Basic flow:
    1. User selects the template group from the select drop down menu.
    2. User clicks the button ‘Delete Group’.
    3. The system displays a pop up to confirm template group deletion.
    4. User clicks button ‘Confirm’.
    5. The system deletes all the templates in that specific template group in the template database.
    6. The system deletes the template group in the template database.
7. Post condition:
    1. The template group and all its associated templates are deleted.
8. Specific Alternative Flows:
    1. RFS Basic flow 4:
        1. IF User clicks button ‘Cancel’, THEN ABORT, ELSE RESUME STEP, ENDIF

**I.05 Add a template**

1. Brief description: Adding a new template for use.
2. Primary Actor:
    1. User

3. Precondition:
    1. Use case ‘Start application’ successful
4. Secondary Actor:
    1. `Template database`
5. Dependency:
    1. None
6. Basic flow:
    1. `User` enters the desired template name in the text field beside the button ‘Add Template’.
    2. `User` clicks the button ‘Add template’.
    3. The system creates the template with the name in the `template database` and links the template to the currently selected template group.
7. Post condition:
    1. The new template is displayed and ready to use in the current template group.
8. Specific Alternative Flows:
    1. None

**I.06 Modify a template’s name**

1. Brief description: The template’s name can be modified after creation.
2. Primary Actor:
    1. `User`
3. Precondition:

1. Use case ‘Start application’ successful
2. Use case ‘Add a template’ successful

4. Secondary Actor:
    1. `Template database`
5. Dependency:
    1. None
6. Basic flow:
    1. `User` replaces the template name in the text field that holds the current template name.
    2. The system displays a header message “Please click 'Save Changes' to commit changes”.
    3. `User` clicks the button ‘Save Changes’ at the bottom of the template.
    4. The system saves the modified template name in the `template database`.
7. Post condition:
    1. The template’s name is changed both front and back end.
8. Specific Alternative Flows:
    1. RFS Basic flow 3:
        1. IF `User` clicks button ‘Discard Changes’, THEN ABORT, ELSE RESUME STEP, ENDIF

**I.07 Customize a button in a template**

1. Brief description: Customize the button component of a template.

2. Primary Actor:
    1. User
3. Precondition:
    1. Use case ‘Start application’ successful
    2. Use case ‘Add a template’ successful
4. Secondary Actor:
    1. Template database
5. Dependency:
    1. None
6. Basic flow:
    1. User clicks on a button that represents a hot key within the desired template.
    2. The system creates a pop up to display the button’s customization options.
    3. User enters a desired text to be the button’s main label.
    4. User enters a desired text for the main content of the button.
    5. User selects the button’s colour through a colour picker.
    6. User clicks the button ‘Save’.
    7. The system updates the interface to reflect the button’s new customization.
    8. The system displays a header message for that template, “Please click 'Save' to commit changes.”
    9. User clicks the button “Save Changes” at the bottom of the template to commit the changes to the template database.
7. Post condition:

1. The button’s customization is saved in the back end and displayed.

8. Specific Alternative Flows:
    1. RFS Basic flow 6:
        1. IF `User` clicks button ‘Discard’, THEN ABORT, ELSE RESUME STEP, ENDIF
    2. RFS Basic flow 9:
        1. IF User clicks button ‘Discard Changes’, THEN ABORT, ELSE RESUME STEP, ENDIF

**I.08 Delete a template**

1. Brief description: Delete an individual template.
2. Primary Actor:
    1. `User`
3. Precondition:
    1. Use case ‘Start application’ successful
    2. Use case ‘Add a template’ successful
4. Secondary Actor:
    1. `Template database`
5. Dependency:
    1. None
6. Basic flow:

1. User navigates to which template group the specific template that is to be deleted is contained.
2. User clicks the button ‘Delete Template’.
3. The system displays a confirmation pop up that specifies the template’s name to be deleted.
4. User clicks button ‘Confirm’.
5. The system deletes the template from template database.

7. Post condition:
    1. The system updates the interface to reflect that the template is removed.
8. Specific Alternative Flows:
    1. RFS Basic flow 4:
        1. IF User clicks button ‘Cancel’, THEN ABORT, ELSE RESUME STEP, ENDIF

**I.09 Transfer a Template to another Template Group**

1. Brief description: Transfer a template to another template group.
2. Primary Actor:
    2. User
3. Precondition:
    3. Use case ‘Start application’ successful
    4. Use case ‘Add a template’ successful
4. Secondary Actor:
    2. Template database

5. Dependency:
    2. None
6. Basic flow:
    1. User navigates to the desired template to be transferred and clicks “Transfer To Group” at the bottom of the template.
    2. The system displays a pop up with valid template groups it can be transferred to.
    3. User selected the desired template group destination from the drop down select menu.
    4. User clicks button “Transfer” to confirm the transfer
    5. The system updates the template group and template relationship in the template database.
7. Post condition:
    2. The system updates the interface to reflect that the template is removed from the current template group.
8. Specific Alternative Flows:
    2. RFS Basic flow 4:
        2. IF User clicks button ‘Close, THEN ABORT, ELSE RESUME STEP, ENDIF

## APPENDIX J – Screenshots of application

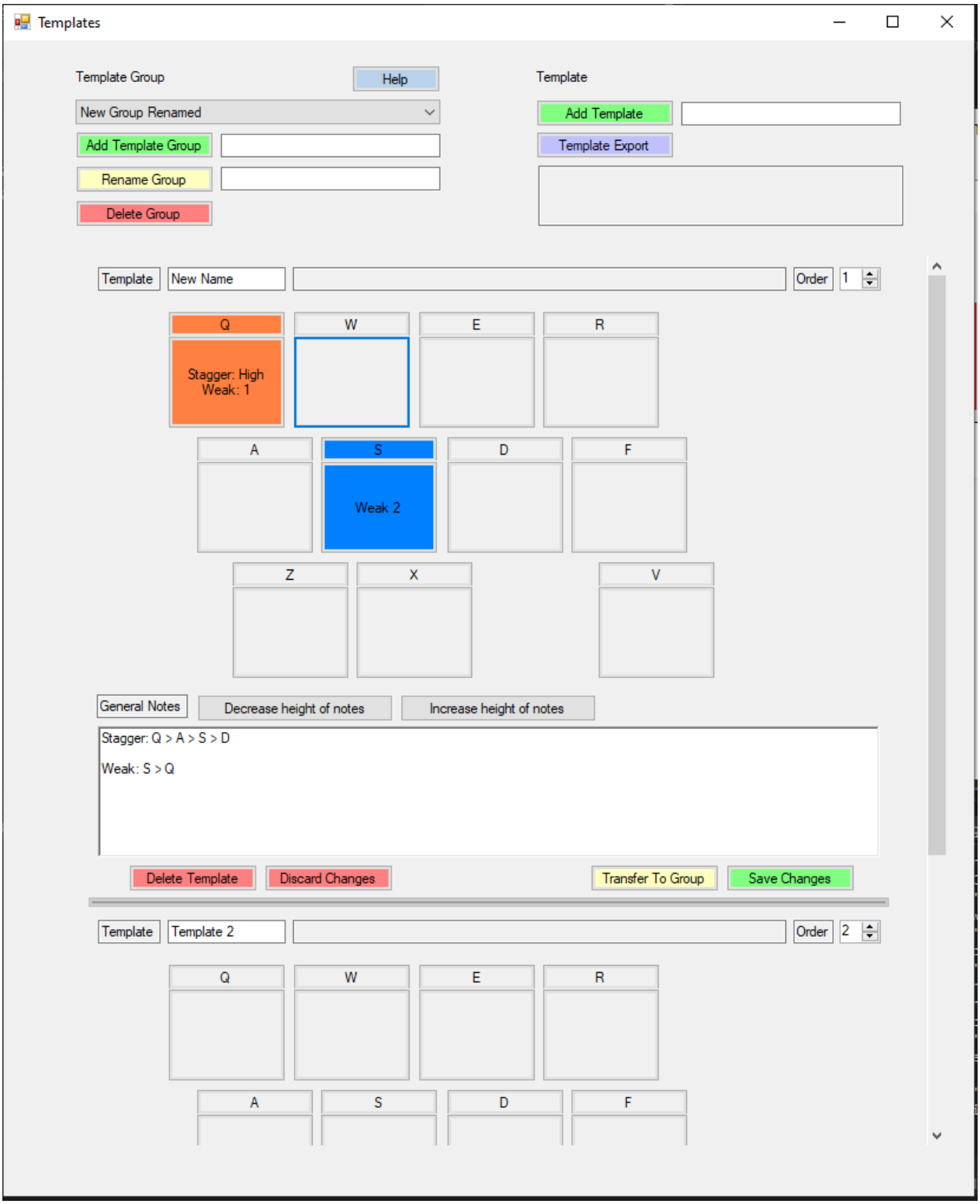


*Figure 11: Screenshot of Main Template Application*

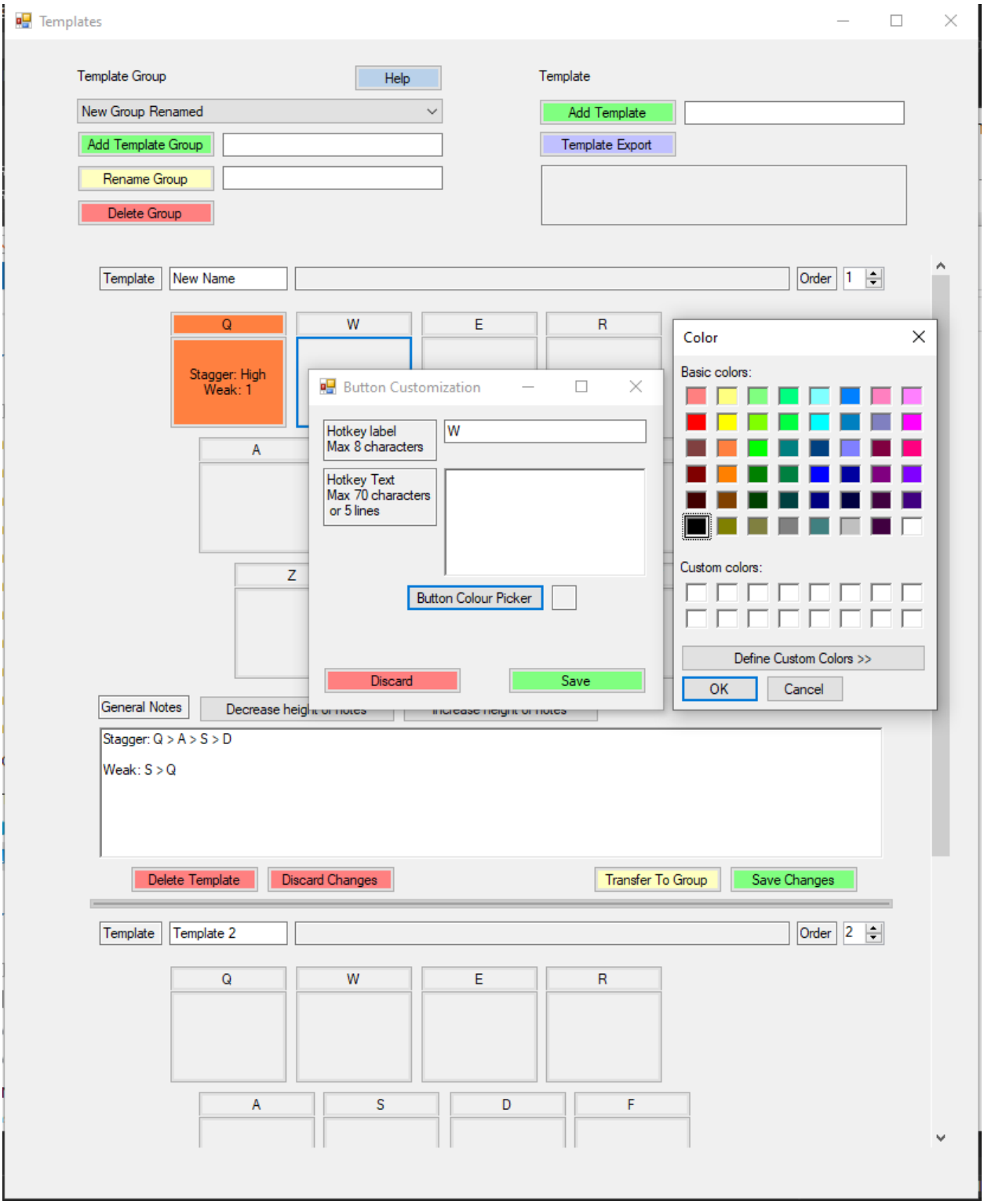


*Figure 12: Screenshot of Individual Button Customization*

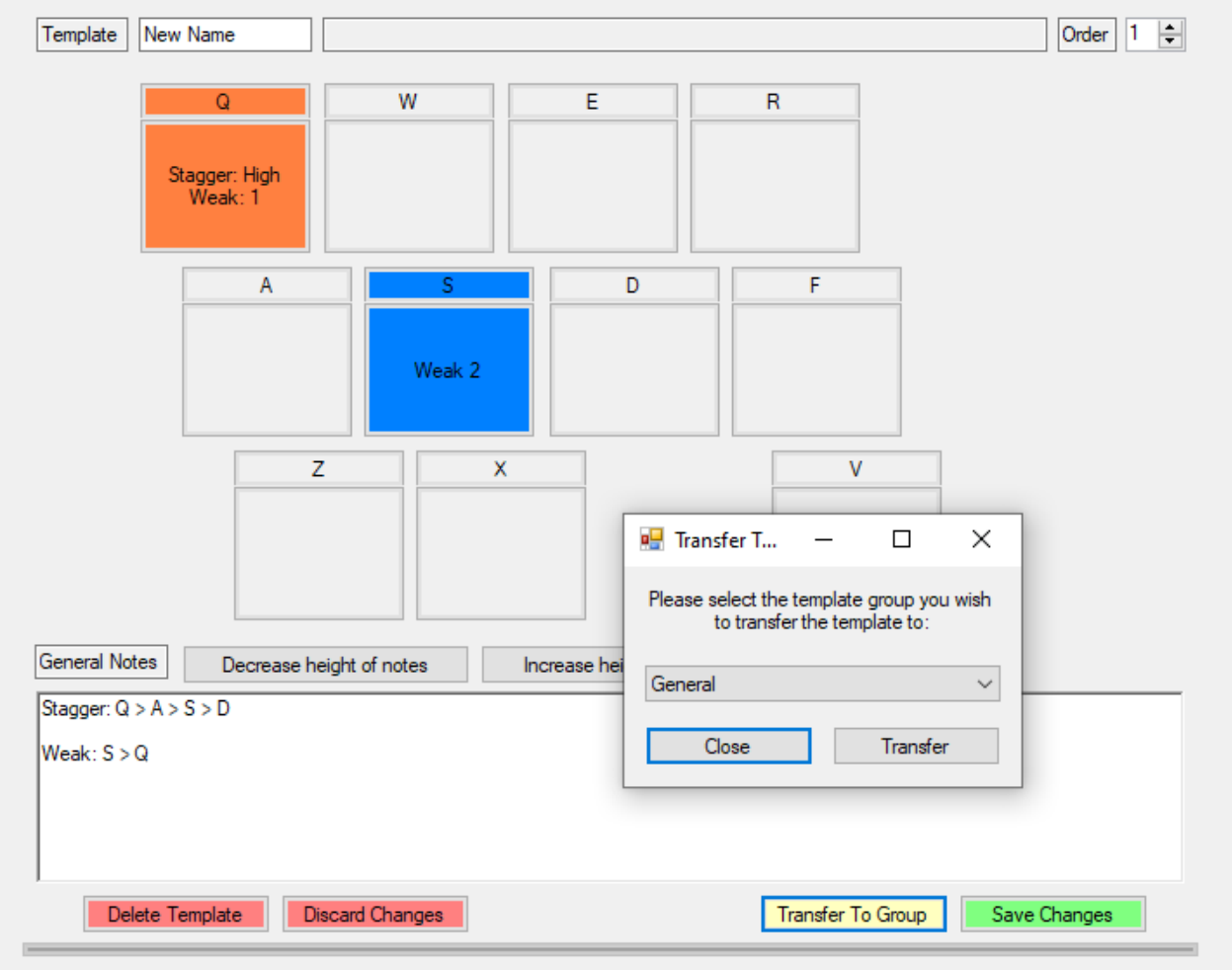


*Figure 13: Screenshot of Transferring Template to another Group*