
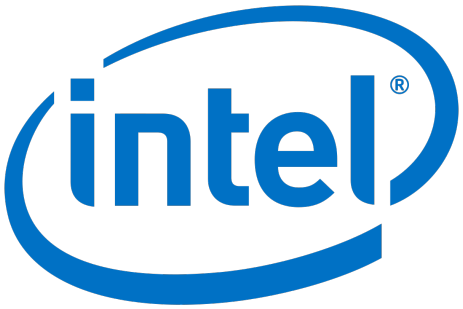


# CS Capstone Winter End-Of-Term Report

March 12, 2020

# Duo Tech: Window Configuration

Prepared for

## Intel

Mike Premi

Prepared by

## Group 66a

## Duo Tech

Sachin Sakthivel

Rosalinda Garcia

Derk Kieft

Matthew Ferchland

**Abstract**

This document describes the progress, problems and solutions we have made over the Winter term. This report will start by giving a brief overview of the project. Next, it discusses our current progress through weekly recaps and future plans. Then, the document describes the problems we have encountered and the possible solutions to those issues. Finally, the document concludes by discussing interesting pieces of code and diagrams of our user interface.

## CONTENTS



## LIST OF FIGURES

# 1 TERMINOLOGY

- **Configuration/Layout/Work-space:** Visible applications that are currently open in a user's environment. (An example of this can be found in figure 1 under the "Preview" label.)
- **SQLite Database:** Local storage database that uses user's hard-drive to store application data.
- **Companion Screen:** Second screen right above the keyboard on the Asus ZenBook Pro Duo.
- **Preview:** This refers to the "preview" label in figure 1 showing the screen orientation of the dual screen setup within the Asus ZenBook Pro Duo.
- **Window Configuration Application:** The application developed by the Duo Tech capstone team that helps manage work-spaces.

# 2 PROJECT OVERVIEW

## 2.1 Introduction

The Asus ZenBook Pro Duo is a new paradigm in laptop computing. While having multiple monitors have become common, the idea of having multiple screens on a mobile device such as a laptop is a relatively new idea. The Asus ZenBook Pro Duo is one of the first multi-screen portable devices on the market and as such, faces some interesting challenges.

About the ZenBook Pro Duo:

- **CPU:** Intel Core I9-9980HK
- **GPU:** NVIDIA GeForce RTX 2060
- **Memory:** 16/32GB 2666Mhz DDR4
- **OS:** Windows 10
- 4K UHD NanoEdge OLED HDR touchscreen display
- 4K UHD NanoEdge OLED HDR ScreenPad Plus companion display

The ZenBook Pro Duo comes with various pre-installed applications and tools to support the companion screen. However, these tools are relatively basic and don't provide enough utility for users. To combat this issue, the goal of the Duo Tech project is to develop an application to enhance the user experience of the ZenBook companion screen as well as innovate tools that might be used in forthcoming dual screen systems.

## 2.2 Project Description

This product originates from the need to develop more software that utilize the capabilities of dual and companion screen systems to enhance the user experience on these new types of devices. While multi-monitor technology has been available for quite some time, the functionality has always been fairly limited. This product will upend the normal use of the second monitor and give the user something truly unique in using the advanced hardware capabilities. The product will rely heavily on the companion monitor technology and focus on the functionality between two touch screen monitors.

We developed our application using Visual Studio's C# .NET Window Forms framework. This product is tailored towards individuals who want to have more freedom in how they manage various windows in their work-space.

The application is used by first taking a screenshot of all the currently used screens and displaying the foreground windows that are present within each screenshot. The user then has the ability to save this configuration to their local SQLite database to be used at a later time. When the user loads a custom configuration, all windows that were previously stored in the database will resize and move based on the specified location and size.

## 3 Current Progress

### 3.1 Week 1

The first week returning after break focused on completing the team critique assignments and getting back into the capstone project. In a meeting with Intel sponsors, we discussed some high level topics including design coordination. A portion of the week was spent on researching the tools we would be using for development including Visual Studio, Windows Forms and C#. A transition/state diagram was also developed to demonstrate how the program components would fit together.

### 3.2 Week 2

In week two, work was started on C# functions for window handlers and processes. This means that we focused our time building the foundational functions for getting metadata information, moving windows and resizing windows. In addition, research was underway in utilizing third party API's to manipulate settings for specific applications such as the Adobe Creative Cloud suite. On the side, there was also work on poster concepts in order to have it completed as soon as possible.

### 3.3 Week 3

Further preliminary work was done on the poster in the third week. Additionally, a local SQLite DB was integrated into the code to allow storage of window handlers. This allowed us to store and retrieve information to facilitate eventual resizing and relocation of windows in the system. Adobe API investigation continued, though it was discovered that the configuration of Adobe products is very restrictive to third party applications. The client agreed that this feature was likely not worth exploring further for the time being. User Interface prototypes were also being developed on paper and then converted to a high resolution version with Adobe XD.

### 3.4 Week 4

Functional prototypes were developed from previous UI concepts and Windows forms were created for the homepage, configuration hub, import, export and the create new configuration page. We also progressed on documentation updates throughout the week and continued work on the poster draft. In addition, this week we previewed the UI with the client and got feedback to help further our development.

### 3.5 Week 5

Application launch errors were debugged this week and error handling was added to the application. We finalized the draft copy of the Engineering Expo poster which was due this week. We also began attempting to implement the change to a system tray application instead of a straight windows form. Lastly, we presented our current functionality to the clients in our weekly meeting and took feedback regarding changes in interface and functionality that might improve the end product.

### 3.6 Week 6

During this week, the UI was redesigned to be closer to the functional prototypes. This included performing an evaluation to find bugs. From these evaluations, bug lists were made and then some small bugs such as labelling were fixed. Additionally, we completed our registration paperwork for the Expo.

### 3.7 Week 7

Additional UI developments were made to clean up the previous week's progress. This included adding rounded buttons and making sure the expected functionality matched the actual user experience. This week we also spent time preparing and practicing our design review presentation. This included creating our slides and practicing our individual parts.

### 3.8 Week 8

This week we had our design review with three other project groups. In addition, we added the ability to open applications by clicking the run config button but had some minor issues with the application launching. We were able to figure out that the bug we were experiencing is due to 32-bit applications not having a main window handler and spent some time attempting to figure out a workaround.

### 3.9 Week 9

During this week, we were still working on fixing the 32-bit application bug. We were able to narrow down the cause by examining the logging info in the console and learned that after the application opened we were unable to access the main window handler value. We realized that this was going to take a while to figure out, so we moved on for the time being, logging it as a known bug. We also started working on the ability to create a shortcut on the desktop that will execute a configuration when ran.

### 3.10 Week 10

During the final week of Winter term, we were able to get the shortcut feature working and added a customized button for it on the UI. We also started writing the end of term report and we each recorded a section of the beta functionality video and combined them all into a single video.

## 4 Future Steps

We are planning to fix the bug that is keeping us from being able to control the size and location of 32-bit apps after we open them. After fixing this issue, our application should be completed but we are going to still keep working on adding extra features, optimizing our code, and polishing our user experience as we come up to the final term and the Spring Engineering Expo.

# 5 PROBLEMS

## 5.1 Foreground Application Detection

One of the first problems that we encountered when developing our application was figuring out what processes would be associated with a particular configuration. In reality, we only want to consider foreground applications that are currently in the user's environment. In this case, foreground applications means any windows that are currently visible to the user and are not considered minimized. However, when looping through each application, C#'s native process function doesn't differentiate between background and foreground windows. This in turn will cause our application to store redundant window handler information for applications that a user may never use in their environment. In addition, background and foreground applications will be launched when a user tries to load a configuration which adds more CPU overhead and produces an unwanted result. Another concern is that the Windows operating system creates suspended Microsoft Store and Microsoft Setting processes which are considered foreground applications even though their windows aren't visible to the user.

## 5.2 Generalizing Monitor Detection

Even though our application was built for the Asus ZenBook Pro Duo, we would like the ability to generalize our software for any multi-monitor layout. However, since our logic is tailored towards the laptop, a lot of our functions fail to perform the proper action for different computer systems. For example, if our window configuration app were to be ran on a single monitor, the app would crash because it was unable to detect any applications from a second monitor. This issue would also occur within the ZenBook Pro Duo if the user decides to disable the companion screen. Another issue that was discovered within our window configuration application is that we fail to properly show the preview images of a particular configuration based on the number of monitors. For example, in figure 1 and 2, our "preview" rectangular image partitions for each monitor is static because we initially only considered displaying the current environment within the ZenBook Pro Duo.

## 5.3 Dots Per Inch (DPI) Variation in Companion Screen

The Asus ZenBook Pro Duo is a unique laptop because it provides the user two screens. The laptop comes with a main screen that is standard across all laptop devices but also comes with a companion screen right above the keyboard. The companion screen is around 1/3 the height of the main screen but it acts like a second monitor. However, the fact that the companion screen is lot smaller than the main screen means that the overall scaling resolution needs to be a lot higher than the main screen. This causes a problem because the main screen and companion screen have differing resolutions which means that capturing the preview of the environment and detecting metadata information for window handlers can be potentially erroneous.

## 5.4 Permission Denial and Application Start-up

One of the core functions of our window configuration app is to load a work-space based on a previously stored user layout. When a user decides to load a configuration, our application will launch all of the processes associated with the work-space, resize the window handlers and finally move the windows to the appropriate location. However, there are several problems that occur when a user tries to load a configuration. The first problem is associated with permissions. Since our application is considered third-party to the Windows operating system, we may not have full permissions for

all applications within a work-space. This means that when we try to launch the executable for a particular application, we may be prevented from opening the process because we lack certain permissions. Similarly, if the user uses a 32-bit application on a 64-bit operating system, our window configuration app will be unable to launch the 32-bit process. The next problem that we encountered is involved with finding the right executable in a user's system. Since our application allows users to share their configuration with other people, we need to be able to find the executable of various applications in any person's computer. However, this isn't a feasible task and may require our application to search the entire file explorer of a computer to find the right executable. This will significantly increase the time to load a configuration and ultimately provide a poor user experience. The final problem involving application start-up is actually moving the windows to the correct location. Since every application and computer has a variable application launch time, it is impossible to predict how long an application will take to start-up. That being said, this leads to the issue where we are unable to determine when we should move the window handler of a process. For example, if we tried to move the window too early, the application would be static because it hasn't been fully launched. Additionally, if we tried to move the window after it launched, it would prevent the user from using their work-space immediately because there would be a slight delay.

### 5.5 User Interface Limitations

The primary problem we faced in developing the UI was in creating a modern UI with rounded buttons and understandable functionality. With creating rounded buttons, there was a problem with the ability of Visual Studio to support 64-bit programs. Inserting custom buttons is also not possible with the settings required to run our 64-bit program. Additionally, we struggled to ensure that the UI was understandable to users outside of our team. We found a number of small bugs that could confuse users such as misleading labels on buttons and inconsistent styling as well as a lack of instructional documentation.

## 6 Solutions

### 6.1 Solution to Foreground Application Detection

Based on some online research, it was discovered that there isn't an explicit way to differentiate between foreground and background applications. However, when looking at the Windows task manager, we noticed that foreground process are usually under the "Apps" section and background applications are placed in the "Background Processes" section. Using this information, we were able to create a solution by examining the styling of all currently opened processes. In other words, any application that has a window styling will be considered a foreground process because it has a visible window that users can see. In addition, we also found that if we use the IsIconic function in C# we can also determine all of the applications that are minimized. Both of these solutions allowed us to loop through each process that is currently running and see if it has a visible window within a user's environment.

### 6.2 Solution to Generalizing Monitor Detection

Generalizing the monitor detection is a fundamental component of our application because we want our software to be used in many different multi-monitor systems. In order to solve the issue of having variable amount of monitors, we decided to allow our database and code to detect the relative coordinates and sizing of each monitor. With that information, we can then properly adjust the window handlers on different screens and ultimately relocate the

applications to the appropriate spot. However, the user interface is currently not reflecting these changes because the ”preview” images are static and don’t dynamically change depending on the size and number of monitors. To solve this problem, the best solution is for our application to match the rectangular image partitions within our preview section with the display screen orientation in the Windows ”Display Settings” panel.

### 6.3 Solution to Dots Per Inch (DPI) Variation in Companion Screen

Solving the different scaling resolutions was a fairly difficult challenge because we need to take into account inflated sizing and erroneous coordinates of different window handlers. For example, the size of a window application on the main screen would be one value but if it was moved to the companion screen it would significantly change. That being said, to solve this problem, we first adjusted the window handler’s coordinates and sizing by either adding or subtracting X amount of DPI depending on what monitor it was currently located on. In other words, based on the size of the monitor, our window configuration software would properly scale the applications to the appropriate values. Another solution that we incorporated within our application was to make it DPI aware. This allowed the window configuration software to adjust the screen resolution based on the scaling of the monitor in order to properly capture each screen for the image preview section.

### 6.4 Solution to Permission Denial and Application Start-up

Our solution to handle the problem of permission denial was suggested by our sponsor who mentioned that we should assume that all users that use our application are launching it in administrator mode. This will allow us to launch applications regardless of the permissions that we currently have. We also solved the issue of moving window handlers when we programmatically launch different processes by using the Process.Start() function in C# which allows you to start an application based on an absolute path, resize the window and relocate the window to a specific location at the same time. In regards to the other two problems, we are still having issues figuring out how to efficiently find an executable in a person’s computer and handling launch issues with 32-bit applications. That being said, our temporary solution for this problem is to give an error message to the user displaying all of the processes our application failed to start, so they can manually launch them.

### 6.5 Solution to User Interface Limitations

In order to add the rounded buttons to the UI, a workaround had to be used. This required us to change the settings of our program in Visual Studio while creating the UI and then reverting these settings to run the program properly. The result of this is a number of errors, however, these are due to the change of settings and still allow for the use of custom buttons. In terms of debugging the UI, we resolved these issues by performing evaluations of the interface and creating bug lists. Based on the lists of bugs, we are continuing to improve and test the UI.

## 7 Interesting Code Segments

The function capture_screens() is used to take a screenshot of all of the monitors that are currently being detected by our window configuration application. We are use the Bitmap and Graphics classes in C# in order to handle DPI scaling and ensure that the screenshot we take is compatible regardless of the size of the monitor. We are also storing the image previews of each configuration within a local directory in order for the user to view the previews of all of their

configurations at a later date.

```
// Loop through all monitors and take a screenshot of each monitor
private void capture_screens()
{
    new_config_page.RefToConfig = this;
    this.Hide();
    new_config_page.screenshotlist.Clear();
    // For each screen take a screenshot of their display
    foreach (var screen in Screen.AllScreens)
    {
        Bitmap screenshot = new Bitmap(screen.Bounds.Width, screen.Bounds.Height
            , System.Drawing.Imaging.PixelFormat.Format32bppArgb);
        Graphics memoryGraphics = Graphics.FromImage(screenshot);
        memoryGraphics.CopyFromScreen(screen.Bounds.X, screen.Bounds.Y, 0, 0,
            screen.Bounds.Size, CopyPixelOperation.SourceCopy);
        new_config_page.screenshotlist.Add(screenshot);
    }
    new_config_page.set_disp();
    this.Show();
}
```

The public_config_Click() function allows the user to create a shortcut to load a window configuration. This is done by creating a shell program that will launch, resize and relocate all of the applications associated with a person's custom work-space. This will simulate a "one-click" button functionality for starting a work-space without using the main window configuration application.

```
private void publish_config_Click(object sender, EventArgs e)
{
    if (cfg_display.SelectedItems.Count > 0)
    {
        for (int i = 0; i < cfg_display.SelectedItems.Count; i++)
        {
            object shDesktop = (object)"Desktop";
            string name = cfg_display.SelectedItems[i].Text;
            WshShell shell = new WshShell();
            string shortcutAddress = (string)shell.SpecialFolders.Item(ref
                shDesktop) + @"\Run_" + name + ".lnk";
            IWshShortcut shortcut = (IWshShortcut)shell.CreateShortcut(
```

```
                    shortcutAddress);
                shortcut.Description = "Run configuration: " + name;
                shortcut.TargetPath = Environment.CurrentDirectory + @"\
                    WindowConfiguration.exe ";
                shortcut.WorkingDirectory = Environment.CurrentDirectory;
                shortcut.Arguments = name;
                shortcut.Save();
            }
        }
        else
        {
            cfg_err_label.Text = "Select a Configuration to create a Desktop
                shortcut for!";
            cfg_err_label.Visible = true;
        }
    }
```

The get_processes() function is one of the core functions of our application. This method will loop through each process and see if its a foreground application by first checking if the window has a styling and checking if its not minimized. After getting all the foreground applications, the function will then construct a window handler object using the metadata information from the process and store it in our local SQLite database.

```
// Get all the foreground processes and display them in a listview in thew new_config
    form
    private int get_processes()
    {
        var processes = Process.GetProcesses().Where(pr => pr.MainWindowHandle !=
            IntPtr.Zero);
        RECT rect = new RECT();
        IntPtr hWnd;
        var process_count = 0;

        new_config_page.clear_process_list_view();
        // Loop through all the currently running processes and get some diagnostic
            information for each window. You can also insert into the DB and move
            the window (uncomment last few statements)
        foreach (var proc in processes)
        {
            if (!string.IsNullOrEmpty(proc.MainWindowTitle))
```

```
{
    // Print out some diagnostic information
    new_config.WindowInfo window = new new_config.WindowInfo();
    hWnd = FindWindow(null, proc.MainWindowTitle);
    GetWindowRect(hWnd, out rect);

    if (IsIconic(hWnd) == false && IsAppWindow(hWnd) && proc.ProcessName
        != "WindowConfiguration")
    {
        // Get mainwindowtitle, location, size and display them in the
            listview for the new_config form
        process_count++;
        string topleftcoord = "(" + rect.Left + "," + rect.Top + "), ";
        string bottomrightcoord = "(" + rect.Right + "," + rect.Bottom +
            ")";
        string location = topleftcoord + bottomrightcoord;
        int width = Math.Abs((rect.Left) - (rect.Right));
        int height = Math.Abs((rect.Top) - (rect.Bottom));
        string size = width.ToString() + " x " + height.ToString();
        String[] row = { proc.MainWindowTitle, location, size };
        new_config_page.set_process_list_view(row);

        // Fill window struct in order to easily send an object to the
            sql insert function
        window.Process_Name = proc.ProcessName.ToString();
        window.Left = rect.Left;
        window.Right = rect.Right;
        window.Top = rect.Top;
        window.Bottom = rect.Bottom;
        window.Width = Math.Abs((rect.Left) - (rect.Right));
        window.Height = Math.Abs((rect.Top) - (rect.Bottom));
        window.Process_Title = proc.MainWindowTitle;
        window.Process_ID = proc.Id;
        window.Exe_Path = proc.MainModule.FileName;
        new_config_page.win_list.Add(window);


    }
}
```

```
            }
            return process_count;
        }
```

## 8 Application Images

Figure 1 is the main homepage of the application where users can browse through their configurations, create new configurations, import, export or load one of their work-spaces. The list-view that displays the configurations are all of the previously stored work-spaces that the user has created. Users can then load a particular layout which will start up the associated applications and move them to the appropriate location. Additionally, users have the ability to share their work-spaces by exporting their database to an external source or use other people's configurations by importing their database.

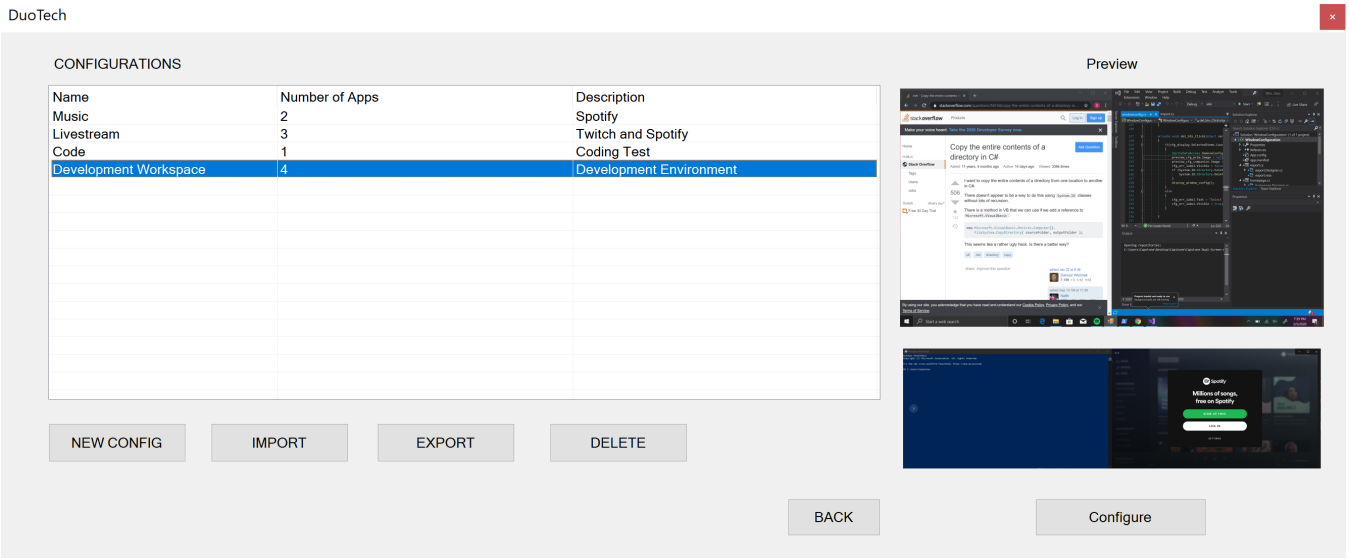


**Figure 1:** Configuration Selection Hub

Figure 2 shows the page that allows users to fill out metadata information for their configuration. For example, users have the ability to give their work-space a name, description, view the processes that are associated with their configuration and a preview of their environment for future reference.

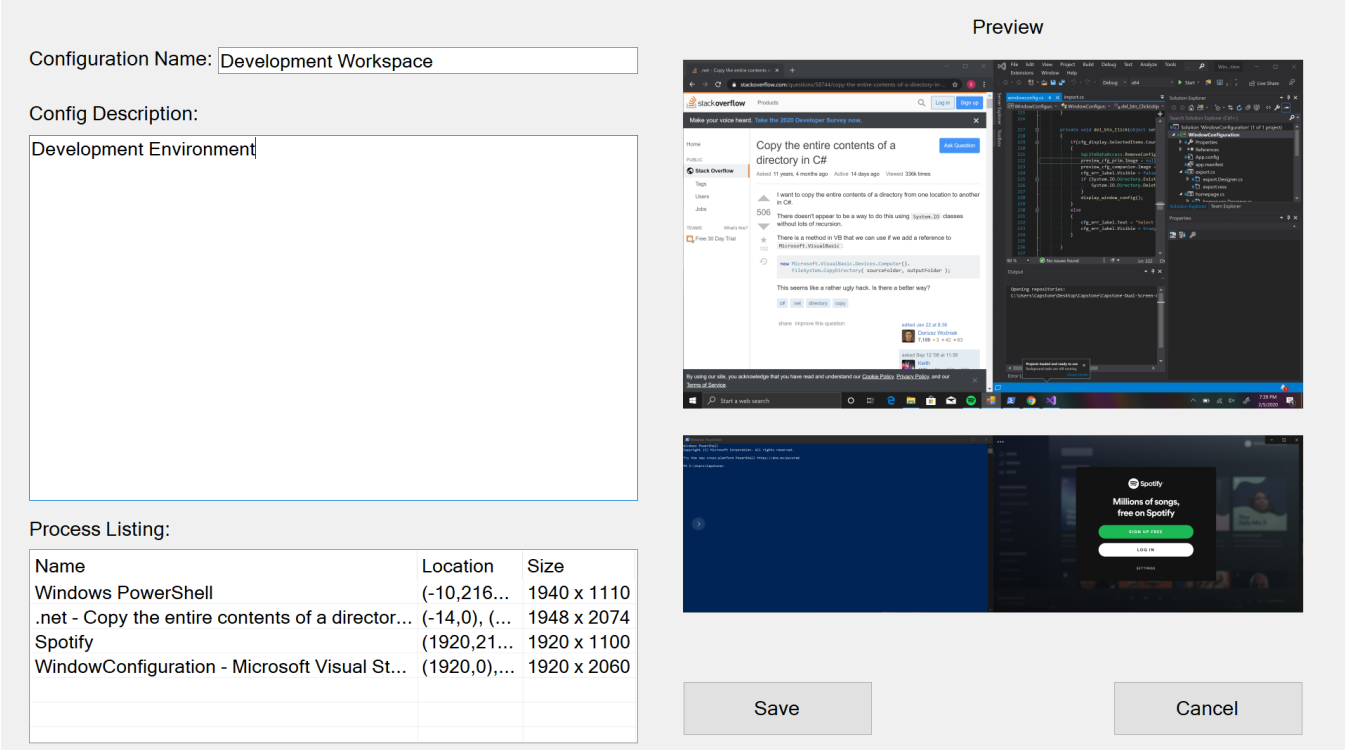


**Figure 2:** New Configuration Page

Figure 3 outlines the high-level architecture of the window configuration application. The application initially takes a configuration as input and allows the user to perform four major operations: save a configuration, export, import and load a configuration.

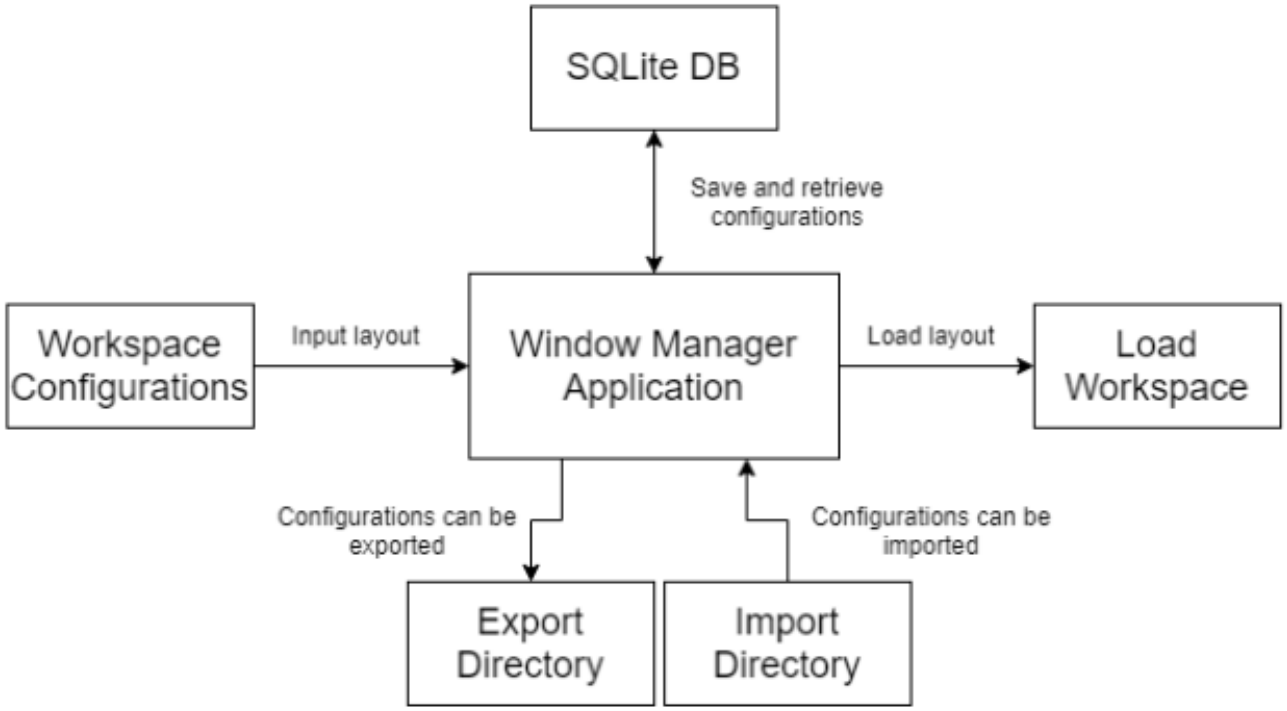


**Figure 3:** Application High-level Architecture

Figure 4 provides an example of how a window configuration is stored within the SQLite database. Each configuration that a user saves will be stored in a relational table where each tuple will be associated with a window handler. Each of these tuples will then have information about the process name, process ID, size of the window handler and the location of the window.

Table: Livestream

New Record | Delete Record

| | ID | Process_ID | Process_Title | Process_Name | Left | Right | Top | Bottom | Width | Height |
|---|---|---|---|---|---|---|---|---|---|---|
| | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | 1 | 4200 | Twitch - Google Chrome | chrome | -16 | 3856 | -16 | 2076 | 3872 | 2092 |
| 2 | 2 | 27168 | Spotify | Spotify | -11 | 3852 | 2149 | 3271 | 3863 | 1122 |
| 3 | 3 | 14368 | WindowConfiguration (Running) - Microsoft Visual Studio | devenv | -16 | 3856 | -16 | 2076 | 3872 | 2092 |

**Figure 4:** Configuration Window Handler Storage

Figure 5 provides an example of how the metadata information is stored within the SQLite database. This table is directly displayed to the user when they are at the configuration selection hub. This table includes information like the name of the configuration, the number of applications associated with the configuration and the description of the work-space.

Table: Table_Description

New Record | Delete Record

| | Name | Num_apps | Description |
|---|---|---|---|
| | Filter | Filter | Filter |
| 1 | Music | 2 | Spotify |
| 2 | Livestream | 3 | Twitch and Spotify |
| 3 | Coding | 2 | Coding |

**Figure 5:** Configuration Metadata