**Movies and Munching (M&M)**

CS261- Algorithm Detail Document



Project Supervisor

Mr. Samyan Qayyum Wahla

Project Members (P67)

|  |  |
| --- | --- |
| Ahmad Arshad | 2020-CS-120 |
| Hamad Ali | 2020-CS-152 |

Department of Computer Science

University of Engineering and Technology, Lahore,Pakistan

Table of Contents

[1 Project Details: 4](#_Toc87026325)

[1.1 Project Title: 4](#_Toc87026326)

[1.2 Problem Statement: 4](#_Toc87026327)

[1.3 Project Description: 4](#_Toc87026328)

[1.4 Required Software’s: 5](#_Toc87026329)

[2 Technical Details: 6](#_Toc87026330)

[2.1 Name of Entity: 6](#_Toc87026331)

[2.2 Attributes of Entity: 6](#_Toc87026332)

[2.3 Sample for Scrapping Source: 7](#_Toc87026333)

[2.3.1 Website: 7](#_Toc87026334)

[2.3.2 Scrapped Data: 7](#_Toc87026335)

[3 Business Demands: 8](#_Toc87026336)

[3.1 Business Need: 8](#_Toc87026337)

[3.2 End User Product: 8](#_Toc87026338)

[3.2.1 App Developers: 8](#_Toc87026339)

[3.2.2 Basic Users/Publics: 8](#_Toc87026340)

[3.3 Motivation for Project: 8](#_Toc87026341)

[3.4 Impacts: 9](#_Toc87026342)

[3.4.1 Impact if project proceeds: 9](#_Toc87026343)

[3.4.2 Impact if project not proceeds: 9](#_Toc87026344)

[4 Algorithms: 10](#_Toc87026345)

[4.1 Sorting Algorithms: 10](#_Toc87026346)

[4.1.1 Selection Sort 10](#_Toc87026347)

[4.1.2 Insertion Sort: 13](#_Toc87026348)

[4.1.3 Bubble Sort: 16](#_Toc87026349)

[4.1.4 Merge Sort 18](#_Toc87026350)

[4.1.5 Quick Sort: 21](#_Toc87026351)

[4.1.6 Counting Sort: 24](#_Toc87026352)

[4.1.7 Bucket Sort: 27](#_Toc87026353)

[4.2 Searching Algorithms: 29](#_Toc87026354)

[4.2.1 Linear Search: 29](#_Toc87026355)

[4.2.2 Binary Search: 30](#_Toc87026356)

[4.2.3 Interpolation Search: 31](#_Toc87026357)

[4.2.4 Exponential Search: 33](#_Toc87026358)

[4.3 Searching Filters: 34](#_Toc87026359)

[4.4 Multi-level Sorting: 34](#_Toc87026360)

[5 User Interphase Detail: 35](#_Toc87026361)

[5.1 Rough UI Look of Pencil Tool: 35](#_Toc87026362)

[5.1.1 Frontier: 35](#_Toc87026363)

[5.2 Main Window: 36](#_Toc87026364)

[5.3 GUI Implementation: 37](#_Toc87026365)

[5.3.1 Frontier: 38](#_Toc87026366)

[5.3.2 Main Window: 39](#_Toc87026367)

[6 Integration: 41](#_Toc87026368)

[6.1 Scrapping: 41](#_Toc87026369)

[6.1.1 IMDB: 41](#_Toc87026370)

[6.1.2 Difficulties: 42](#_Toc87026371)

[6.1.3 Solution: 46](#_Toc87026372)

[6.2 Data Storage: 46](#_Toc87026373)

[6.2.1 Excel: 46](#_Toc87026374)

[6.3 UI Integration: 47](#_Toc87026375)

[6.3.1 Requirement: 47](#_Toc87026376)

[6.3.2 Pyqt Designer: 48](#_Toc87026377)

[6.3.3 Rough Look: 49](#_Toc87026378)

[6.3.4 Designer Implementation: 50](#_Toc87026379)

# Project Details:

## Project Title:

**M&M (Movies and Munching)**

## Problem Statement:

As it's the period where movies are significant wellspring of diversion and they assume significant parts in our day to day existence through their story and morals. Films urge us to make a move. Our favorite characters, superheroes, show us life illustrations. They give us thoughts and motivation to thoroughly take care of the better rather than simply lounging around, trusting that things will turn out well for them.

But everyone has their own taste in what they want to see because movies come in numerous varieties such as Action, Comedy, Drama, Romance, and Thriller etc.

To overcome these issues, we will aggregate a tremendous arrangement of data about films from various websites. In gathering a lot of Information web scraping is one of the most outstanding alternative, it accumulates and plans data significantly faster which will give us amazing help.

## Project Description:

Data Scraping is the major Goal of our Project. We have to gather a large amount of data through scrapping. Our software will scrap movies of multiple types. We are going to scrap about 1 million of movies from various websites like IMDB, hot star etc. and then we will manipulate it through various processes. Scrapping tasks will provide us with the following options: Pause, Start, Resume and Stop with the progress bar showing the progress of tasks/ number of entities scrapped.

We will provide GUI based software. Only particular users will have access to the system. Data will be shown in the table in a Sorted manner.

Our system will provide multiple sorting as per user choice. Our framework will give various arranging according to client choice. Users can arrange data in ascending or descending order or row and column wise. Clients can likewise sort by Movie Name, Director Name, Movie rating and so on Clients can likewise sort films with his very own selected Algorithm. The client additionally has a choice to look through any class of film. He can sort the information in different kinds relying on the particular section through different columns. The software additionally shows the time devoured by the framework to sort the data. There are numerous entities of each film on the table. The user can sort the data with respect to any of the entities. He also has an option of multilevel sorting in it. He can sort data with respect to every column.

In addition, to make it more effective, a legitimate graphical user interface (GUI) will be introduced in which the client will have the accompanying features:

* Sorting of data along any entity of the movie.
* Sort the Data According to multiple Sorting Algorithm
* Searching based on Each Column
* Multi-Level sorting of movies.
* Time for the sorting in milliseconds.
* Progress bar showing the status of the data scraping.
* Searching with multiple algorithms

## Required Software’s:

We are using the following working environment while completing all the phases of the projects:

* + MS Word
  + Pencil Tool
  + Spyder, Jupyter, VS code
  + PYQT
  + BS4(Beautiful Soap)
  + Chrome Web Driver
  + Pichon
  + Annaconda CMD.exe Prompt

# Technical Details:

## Name of Entity:

According to the given requirement our system contains the following entities:

* + Movie name
  + Director name
  + Movie Type
  + Movie Year
  + Movie Actor
  + Movie Rating
  + Duration

These are some of the entity that are well known for any movie and about which the users are interested in.

The all entities of the movies are provided in a well-managed form. So the users have an easy access to that. Every user can find any kind of movie of his choice by searching through any of the following entity of his taste and choice.

## Attributes of Entity:

|  |  |  |
| --- | --- | --- |
| *Name* | *Data Type* | *Description* |
| Movie Name | String | An identifying name of the movie |
| Director Name | String | Name of the Director of the movie who determines the creative vision of that movie |
| Movie Type | String | Determine the type of movie . |
| Movie Year | Date | The year in which the movie has been released |
| Movie Actor | String | List of the cast of the movie |
| Movie Rating | Integer | Ratings of the movie in points |
| Duration | String | Total time of the movie |

## Sample for Scrapping Source:

### Website:



**Fig 2.3.1:** IMDB website movie screenshot with details

### Scrapped Data:



**Fig 2.3.2:** Scrapped Data from CSV to Table

# Business Demands:

## Business Need:

As we know mostly people consume their free time by watching movies or TV shows but the biggest problem they faced is, they couldn’t get all movies or just movies name in a single website or software This project will be very simple. Its main purpose is to create a connection with our customers that movies are showing at what times with link of each movie along with reviews. The design will feature a list of current movies and will be small enough to quickly load on most home computers.

## End User Product:

We have scrapped a vast amount of movies data from multiple sources. Every job has a purpose and has a healthy use for user end. We should think of the top users that can be helped by this movie data scrapping project:

### App Developers:

We People who are dealing with movies app development. They must be in need of movie scrapping projects that would affiliate them for gathering a vast amount of data to facilitate their app development. It can create a great ease for them as they do not have to scrap data their selves.

### Basic Users/Publics:

This will also facilitate the normal public. They can use this system to view a lot of movies to keep themselves up-to-date. Here they can search any kind of any movie and have a list of information for that movie. They can make it really quick for them to choose and find a lots of title of movies of their interest by just searching the movie category. Every person has their favorite actors or directors and they keep an interest for their work. They want to be their viewers so they can find those particulars movies by just searching through their favorite actor’s and director’s names.

## Motivation for Project:

As it's the period where movies are significant wellspring of diversion and they assume significant parts in our day-to-day existence through their story and morals. Films urge us to make a move. Our favorite characters, superheroes, show us life illustrations. They give us thoughts and motivation to thoroughly take care of the better rather than simply lounging around, trusting that things will turn out well for them. But Everyone has their own taste in what they want to see because movies come in numerous varieties such as Action, Comedy, Drama, Romance, Thriller etc. To overcome these issues, we will aggregate a tremendous arrangement of data about films from various websites. In gathering a lot of Information web scraping is one of the most outstanding alternatives, it accumulates and plans data significantly faster which will give us amazing help.

## Impacts:

### Impact if project proceeds:

Many purposes like app development on movies has a large demand for such scrapping system. This system will be very efficient for most of the clients. On the other hand, there is high public demand and interest was in movies. This platform provides them an environment where they can find any movies based on their taste of watch by using any of the entities which they have used.

### Impact if project not proceeds:

As data scraping is broadly use we need to make our undertaking more proficient and quick to run it in industry in the event that it gets reach at we need to give them most dependable framework however on the off chance that we neglect to do as such and project not proceeds likely associations won't confide in us once more.

# Algorithms:

Algorithm is known as the procedure in which we design a step by step procedures to get a desire output. It can be done in various ways and various types. In Computer Science while writing an algorithm for any task we do not take care of the programming language. We write it in simple language and implement it through our codes. In this project we have to use two types of algorithms.

## Sorting Algorithms:

Sorting Algorithm are known as the algorithm in which we sort all the element of a given data structures in a specific order and return the result. In this project we have to use the following sorting algorithms:

### Selection Sort

|  |  |
| --- | --- |
| Description | Selection Sort is a comparison sorting method that is used to sort a random array of elements in ascending order. The comparison does not need a large amount of additional space. The temporal variable simply takes one more memory space. Thus it is referred to as "in-place sorting."  The first list comprises things that have been sorted, whereas the second list contains items that have not been sorted. First Array is set empty by default, on the other hand all the elements are placed in Unsorted Array. The minimal value is then found in the unsorted list and entered into the sorted list. This procedure is continued until all of the data have been sorted and compared. Selection sort is named for the fact that it constantly chooses the next-smallest element and switches it into the correct position.  Following steps are taken to sort Array through Selection Sort:   1. We look for the smallest element in the array starting with the first element and element on the first position is replaced with this element but if smallest element is placed on first position then it will remain on its place 2. Then we go to the second location and seek for the smallest element in the subarray, beginning at index 1 and going all the way to the final index. 3. Then we replace the second smallest element with the element placed on second index 4. This process is continued until the array has been sorted fully.   Selection Sort is not a stable algorithm because it might recheck the element which are same and may change their order. Selection sort is also non-adaptable because it does not get effected by the order of elements simply if half of the array is sorted it still check out each elements and no early breakout occurs. |
| Pseudo Code | Selection\_Sort ()   1. sort list : array of items 2. n : size of list 3. for I = 1 to n-1 4. min=i 5. For j = i+1 to n 6. If list[ j ] < list[ min ] then 7. Min=j 8. If indexMin!=I then 9. Swap list[ min] and list[ i]   End if |
| Code In Python | def selectionSort(Arr):  for i in range(len(Arr)-1):  min=i  for j in range(i+1,len(Arr)):  if(Arr[j]<Arr[min]):  min=j  Arr[i],Arr[min]=Arr[min],Arr[i]    return Arr |
| Time Complexity Analysis | |  |  |  | | --- | --- | --- | | 1. for I = 1 to n-1 2. min=i 3. For j = i+1 to n 4. If list[ j ] < list[ min ] then 5. Min=j 6. Swap list[ min] and list[ i] | Cost | Times | | C1 | n | | C2 | n-1 | | C3 | Σn - 1j = 1(tj) | | C4 | Σn - 1j = 1 (tj - 1) | | C5 | Σn - 1j = 1 (tj - 1) | | C6 | n-1 |   Total Time:  T(n) = O n^2 |
| Proof of Correctness | Following three steps will explain Proof of Correctness:-   1. Initialization:-   Being a selection sort loop that passes over all of the array’s indexes. Before starting the first iteration one element from the array will be selected and is the smallest element for the outer loop of whole array. Considering that whole array contain only one element, the array is sorted already for the first index   1. Maintenance:-   Now we'll show how loop varient is kept across the loop's iterations. Because the first use of min has to look for every element in the array, the body of the loop will run n times. The total number of elements in the array is indicated by the letter n. Now that the first elements have been sorted, the loop will run till n index starting from 1and swap the elements if required. The loop will continue checking elements n-1 to n for the following iteration.   1. Termination:-   At the termination of the loop iteration equals n. We'll receive a sorted array after comparing the array of smallest elements with minimum. Sorted array will be produced, and the loop will come to an end. By passing every iteration of loop invariant, the algorithm is shown to be valid. |
|  |  |
| Three Strengths | 1. It does not require much space 2. It is well for small set of data 3. It is an in-place algorithm |
| Three Weaknesses | 1. It shows poor efficiency with huge set of data 2. It takes O(n^2) time for sorting n digit data set 3. Other algorithms performed well then this. |
| Dry Run |  |

### Insertion Sort:

|  |  |
| --- | --- |
| Description | Insertion Sort is a sorting algorithm that works by continually putting the next element into the final data structure in the right order with regard to previously inserted elements.  Insertion Sort is a simply working algorithm. It uses sequentially searching technique. First of all, it compares first two element of an array if both are in sorted manner or in ascending order than it consider the first element in sorted sub-array and then move forward to check second and third element if they are in ascending order it consider second element in sorted sub-array but if both are not in ascending order it swaps their position so they become sorted and also check all elements present in sorted sub-list and if find any element greater then it keeps on swapping. This whole process continues until array get sorted.  Insertion sort works in the same manner that many individuals sort a deck of cards. We begin with an empty left hand and the cards on the table face down. We then take one card from the table at a time and place it in the corresponding spot in the left hand. To determine a card's proper location, we compare it to each of the other cards in the hand, from right to left. The cards in the left hand are always sorted, and they were the top cards of the table pile when they were first dealt.  Following are some of the key features of Insertion Sort:   1. It works efficiently for smaller data sets only 2. Insertion Sort is adaptive, which means that if a partly sorted array is supplied as input, it decreases the overall number of steps, making it more efficient. 3. It is a stable sorting strategy since it does not alter the relative rank of equal components. |
| Pseudo Code | Insertion\_Sort(A)  1.**for** j=2 to A.length  2. key = A[j]  3. i = j-1  4. **while** i > 0 and A[i] > key  5. A[i + 1] = A[i]  6. i = i -1  7. A[ i + 1 ] = key |
| Code In Python | def Insertion\_Sort(A):  for j in range(1,len(A)):  key = A[j]  i = j  while i>0 and A[j-1]>key:  A[i]=A[i-1]  i = i-1  A[i]=key |
| Time Complexity Analysis | |  |  |  | | --- | --- | --- | | **Pseudo Code** | **Cost** | **times** | | 1.**for** j=2 to A.length  2. key = A[j]  3. sequence A[1…j-1]  4. i = j-1  5. **while** i > 0 and A[i] > key  6. A[i + 1] = A[i]  7. i = i -1  8. A[ i + 1 ] = key | C1 | n | | C2 | n-1 | | C3 | n-1 | | C4 | n-1 | | C5 | Σn - 1j = 1(tj) | | C6 | Σn - 1j = 1 (tj - 1) | | C7 | Σn - 1j = 1 (tj - 1) | | C8 | n-1 |     Total Time:  T(n) = C1 \* n + ( C2 + C3 + C4) \* ( n - 1 ) + C5 \* Σn - 1j = 1( t j ) + ( C6 + C7 ) \* Σn - 1j = 1( t j ) + C8 \* ( n - 1 )  Further Simplifications:  T(n) = O( n2 ) |
| Proof of Correctness | Following three steps will explain Proof of Correctness:-   * **Initialization:**   We start by showing that the loop invariant holds before the first loop iteration, when j = 2. The subarray A[1 ,.., j -1], therefore, consists of just the single element A[1], which is in fact the original element in A[1]. Moreover, this subarray is sorted (trivially, of course), which shows that the loop invariant holds prior to the first iteration of the loop   * **Maintenance:**   Next, we tackle the second property: showing that each iteration maintains the loop invariant. Informally, the body of the for loop works by moving A[j – 1], A[j -2], A[j – 3], and so on by one position to the right until it finds the proper position for A[ j ]   * **Termination:**   Finally, we examine what happens when the loop terminates. The condition causing the for loop to terminate is that j > A:length = n. Because each loop iteration increases j by 1Observing that the subarray A[1 ,.., n] is the entire array, we conclude that the entire array is sorted. Hence, the algorithm is correct |
| Three Strengths | 1. It is well for only a small set of data. 2. It is a stable Sorting Algorithm 3. It is simple and easy to write |
| Three Weaknesses | 1. It required n^2 steps for sorting N elements 2. It is a stable Sorting Algorithm 3. It doesn’t deal well with huge set of data |
| Dry Run |  |

### Bubble Sort:

|  |  |
| --- | --- |
| Description | Bubble sort is a sorting algorithm that works by walking through lists of elements that need to be sorted repeatedly and compare pair of elements which are placed adjacently and swap them if they are in the incorrect order. This operation is continued until no swaps are necessary, indicating that the array has been sorted. Basically, bubble sort is all about swapping two elements until our array is sorted. Large values are always sorted first in bubble sort.  Bubble sort gets its name from how smaller and bigger pieces "bubble" to the top of a dataset. For the contrary reason, this technique is also known as the sinking sort: some of the components sink to the bottom of the collection.  Bubble Sort compares each pair of array elements until the whole array is sorted in ascending order. This might lead to some complications, such as what if the array doesn't require any more swapping since all of the elements are already in ascending order. To get rid of this problem, we utilize one flag variable switched to test whether any swaps have occurred. It will exit the loop if no swap has happened so, the array stops processing to be sorted. Another problem is that the highest values settle near the end of the array after each iteration. As a result, the following iteration does not need to contain components that have already been sorted. In order to prevent previously sorted data, we limit the inner loop in our implementation.  Because the technique is simple to comprehend and implement, bubble sort is a good way to teach novice programmers how to sort data sets. It just only a few lines of code, is simple to understand, and can be used anywhere in your software. For larger data sets, however, it is exceedingly inefficient. |
| Pseudo Code | Bubble\_Sort(A)   1. For I =1 to A.lentgh-1 2. For j = A.length downto i+1 3. If A[j] < A[j -1]   Exchange A[ j] with A[j-1] |
| Code In Python | def bubbleSort(A):  for i in range(len(A)):  for j in range(0, len(A)-i-1):  if A[j] > A[j+1] :  A[j], A[j+1] = A[j+1], A[j] |
| Time Complexity Analysis | |  |  |  | | --- | --- | --- | | Pseudo Code | cost | Time | | Bubble\_Sort(A)   1. For I =1 to A.lentgh-1 2. For j = A.length downto i+1 3. If A[j] < A[j -1] 4. Exchange A[ j] with A[j-1] | C1 | 2n | | C2 | (n-1)+(n(n-1))/2 | | C3 | 3(n(n-1))/2 | | C4 | n(n-1) |   Total Time:  T(n) = O ( n^2) |
| Proof of Correctness |  |
| Three Strengths | 1. It is very easy to implement. 2. Don’t require large amount of memory. 3. Easily understandable. |
| Three Weaknesses | 1. It has O(n^2) time complexity. 2. Slow for large no of inputs |
| Dry Run |  |

### Merge Sort

|  |  |
| --- | --- |
| Description | Merge sort is comparison-based algorithm and is the most efficient one. It basically works on the principle of Divide-and-Conquer. It basically involves the following three steps:   1. **Divide:**   It divide the larger problem into the number of smaller sub-problems. In merge sort array split recursively itself in smaller sub-problems till it reaches the base case   1. **Conquer:**   In conquer, smaller sub-problems are recursively solved through comparison.   1. **Combine:**   In this step **merge()** function is used to put sub-arrays back together in the same order as they were divided.  In merge sort array is divided into two sub-arrays. Like if the original array contained four elements, merge sort will divide it into two subarrays, each with two elements. However, splitting the original array into two smaller subarrays does not result in sorted array. As a result, we'll divide these subarrays down even more, until we have many subarrays each containing a single element. Now array with single element is consider to be sorted and we get our base case. Then, one by one, we must merge all of these sorted subarrays until we get single sorted array.  There are two ways to use Merge Sorting Algorithm:   1. **Top-Down Approach** 2. **Bottom-Up Approach**   In Top-Down approach we use recursion to divide array into its halves until we get a single element at the end and in Bottom-Up approach we use iteration to merge single elements into a one sorted array here we compare two elements and merge them and we keep using this until we get a single sorted array in the end.  It is a stable sorting algorithm because adjacent elements in an array tends to keep their original position. It is not an in-place sorting technique because its space complexity is O(n). It is also an External sorting technique. Merge sort is used if the input size exceeds the RAM size and is ideal for large datasets. |
|  |  |
| Pseudo Code | Merge\_sort(A,p,r)   1. If p<r 2. q = [(p + r)/2 ] 3. Merge\_sort(A,p,q) 4. Merge\_sort(A,q+1,r) 5. Merge(A,p,q,r) |
| Code in Python | def merge(arr,leftArr,rightArr):  i = 0  j = 0  k = 0  # Merging  while i < len(leftArr) and j < len(rightArr):  if leftArr[i] < rightArr[j]:  arr[k] = leftArr[i]  i =i+ 1  else:  arr[k] = rightArr[j]  j =j+ 1  k = k + 1  #remaing one merged in the array  while i < len(leftArr):  arr[k] = leftArr[i]  i += 1  k += 1  while j < len(rightArr):  arr[k] = rightArr[j]  j += 1  k += 1  def mergeSort(arr):  if len(arr) > 1:  m = len(arr)//2    #slicing the arrays into two parts  leftArr = arr[:m] #start from 0 and end to mid  rightArr = arr[m:] # start from mid to end    mergeSort(leftArr)  mergeSort(rightArr)  merge(arr,leftArr,rightArr) |
| Time Complexity Analysis | Analysis:  As we know, single step operation to find out the middle of any sub-Array O(1)  To merge the subarrays made by dividing the original array of n densest , a running time of O(n)  Total time:  Hence the total for merge sort function will become n(log n +1) which gives time complexity of O(n\*logn)  T(n) = O(nlogn) |
| Proof of Correctness | Following three steps will explain Proof of Correctness:-  Loop Invariant:  The Array is sorted and the (k-p+1) is the smallest elements of Left  Array and the right array sorted in increasing order.   * **Initialization:**   At k=p, the Array contains a single element which is trivially sorted. And A[p] is the smallest element of Left array and the right array   * **Maintenance:**   Let we assume here the merge satisfy the loop invariant property until k. then (k-p+1) is the smallest elements of Left array and right array which is already sorted in the array A. Next value to be inserted is the smallest one remaining in Left array and right array. then this value is large than those previously inserted in the array A.   * **Termination Step:**   The loop terminates when k = r , this will be done when r – p +1 elements have been inserted in the array A. which means all elements are sorted |
| Three Strengths | 1. It is a stable sort algorithm. 2. It has a consistent running time 3. Efficiently works for large data sets |
| Three Weakness | 1. Additional storage capacity for subarrays 2. For tiny arrays, it is sluggish. 3. Even if the array has already been sorted, the algorithm performs the execution. |
| Dry Run |  |

### Quick Sort:

|  |  |
| --- | --- |
| **Quick Sort** |  |
| Description | Quicksort is a common sorting algorithm that, in comparison to other sorting algorithms, is typically quicker in reality. It divides a large array into two sub-arrays and uses a divide-and-conquer approach to swiftly sort data elements. Quicksort is a common sorting algorithm that, in comparison to other sorting algorithms, is typically quicker in reality. A huge array is divided into two arrays, one of which contains values lower than the provided value, say pivot, on which the partition is based, and the other of which contains values larger than the pivot value. In Quick Sort, the division of an array of elements may be done in any ratio, not just half.  In Quicksort, Pivot is the basis of its working we will choose highest value in the array as pivot and then take two variables one on left and other on right side of pivot as left pointing the low and right pointing the high index respectively. If value at left index is less and value at right index is greater pivot will move to right otherwise left. If both step doesn’t match it will swap left and right. If left is greater than or equal to right, then new pivot will be point they met. Further, we will use quicksort right and left partition recursively.  Quick Sort is a type of internal algorithm and also known by the name of **“partition exchange sort”.** It’s more efficient with smaller datasets only but it does not require any extra space and works way more efficient than merge sort in smaller datasets. Quick Sort is an unstable algorithm but we can make it a stable algorithm by changing few things in code and it is preferred to use Quick Sort for Arrays. |
| Pseudo Code | QuickSort(A, p , r)   1. if p < r 2. q = Partition( A, p , r) 3. QucikSort(A, p , q-1) 4. QuickSort(A, q+1 , r)   Partition(A , p , r)   1. X= A[r] 2. I = p-1 3. For j = p to r-1 4. If A[j]<= x 5. I = I +1 6. Exchange A[ I ] with A[ j ] 7. Exchange A[ i+1 ] with A[ r ] 8. Return i+1 |
| Code in Python | def quickSort(arr, low, high):  if (low < high):  pi = part(arr, low, high) #calling function of partition  quickSort(arr, low, pi – 1) #recursion  quickSort(arr, pi + 1, high)#recursion  def part (arr, low, high):  pivot = arr[high]  I = (low – 1)  for j in range(low,high):  if (arr[j] < pivot):  i=i+1  arr[i],arr[j]=arr[j],arr[i] #swapping    arr[I + 1],arr[high]=arr[high],arr[i+1] #swapping  return (I + 1) #returning the index of sorted pivot |
| Time Complexity Analysis | **Analysis:**  The partitioning technique produces one subproblem with n-1 elements and one with 0 elements, which is the worst-case outcome for quicksort. Assume that each recursive call results in uneven partitioning. The partitioning takes O(n) time to complete. T(0) = O(1) and the recurrence for the running time is because the recursive call on an array of size 0 just returns.  Total Time:  T(n) = T(n-1) + T(0) + O(n)  =T(n-1)+O(n)  if we sum the costs incurred at each level of the recursion, we get an arithmetic series, which evaluates to O(n^2)  T(n) = O(n^2) |
| Proof of Correctness | Following 3 steps can justify Proof of Correctness: -   * + **Initialization:**   Before the primary iteration of the loop, i = p - one and j = p. as a result of no values lie between p and that i and no values lie between i + 1 and j - 1, the first 2 conditions of the loop invariant are trivially satisfied. The line 1 assignment satisfies the third condition   * + **Maintenance:**   We tend to contemplate two cases, reckoning on the result of the take a look at in line 4. once A[ j ] > x; the only action within the loop is to increment j . when j is incremented, condition two holds for A[j – 1] and {every one} different entries stay unchanged. when A [j ] <=x; the loop increments i, swaps A[ i ] and A [j ], so increments j.   * + **Termination:**   At termination, j = r. Therefore, every entry in the array is in one amongst the 3 sets delineated by the invariant, and that we have divided the values in the array into three sets: those under or up to x, those bigger than x, and a singleton set containing x. |
| Three Strengths | 1. No additional storage requires. 2. It has a high term of efficiency for huge list of items. 3. It has short inner loop |
| Three Weakness | 1. It has O(n^2) time complexity. 2. It is recursive. 3. It is place .That no additional storage is required. |
| Dry Run |  |

### Counting Sort:

|  |  |
| --- | --- |
| Description | The Counting Sort Algorithm is a fast sorting algorithm for elements in a given range. When the range of keys is short and there are duplicate keys, this sorting strategy is used. Sorting Algorithms that compare data in numerous passes vary from counting sorts. They function by constructing an array of counters the size of the list's biggest integer hence, the keys must be integers or data that can be easily converted to numbers. It is an integer-based sorting algorithm, as opposed to comparison-based algorithms. It is the only sort which performs in linear time.  Iterating over the input, counting the number of times each item appears, and utilizing those counts to determine each item's index in the final, sorted array is how counting sort works. In counting sort, we need an additional array to sort all the elements of array. It is a stable sort because order of element of final sorted array is same as an input array. It is also a quite fast algorithm then other but it is suitable for smaller data sets only and is not suitable for alphabets or string values. Because it works on counting principle and we cannot count the string. It is efficient for non-negative algorithms only it does not work on negative integers. |
|  |  |
| Pseudo Code | Counting\_Sort(A,B,k)   1. Let C[0..k] 2. **for** i =0 to k 3. C[i] = 0 4. **for** j = 1 to A.length 5. C[ A[ j] ] = C[ A [ j ] ] + 1 6. **for** I = 1 to k 7. C[ i ] = C[i] + C{ i+ 1] 8. **for** j = A.length down to 1 9. B[ C [ A [j] ] ] = A[ j ] 10. C[ A [j] ] = C[ A [j] ] - 1 |
| Code in Python | def Counting\_Sort(A):  n = len(A)  C = [0] \* n  B = [0] \* 10  for i in range(0, n):  B[A[i]] += 1  for i in range(1, 10):  B[i] += B[i - 1]  i = n - 1  while i >= 0:  C[B[A[i]] - 1] = A[i]  B[A[i]] -= 1  i -= 1  for i in range(0, n):  A[i] = C[i] |
| Time Complexity Analysis | |  |  |  | | --- | --- | --- | | **Pseudo Code** | **Cost** | **times** | | Counting\_Sort(A,B,k)   1. Let C[0..k] 2. **for** i =0 to k 3. C[i] = 0 4. **for** j = 1 to A.length 5. C[ A[ j] ] = C[ A [ j ] ] + 1 6. **for** I = 1 to k 7. C[ i ] = C[i] + C{ i+ 1] 8. **for** j = A.length down to 1 9. B[ C [ A [j] ] ] = A[ j ]   C[ A [j] ] = C[ A [j] ] - 1 | C1 |  | | C2 | n-1 | | C3 | n-1 | | C4 | n-1 | |  | |   Let we divide the pseudo code into 5 steps:  In step one : it takes constant time  In Step two: this for loop executed for k times so it takes O(k)  In Step three: this for loop executed for n times so it takes O(n)  In Step four: this for loop executed for k times so it takes O(k)  In Step five: this for loop executed for n times so it takes O(n)  **Total Time**  So the the Total time will be T(n) = O( k + n) |
| Proof of Correctness | **Initialized/established:**  At very start the Count array initialized by 0  **Maintained:**  Lines 4–5 contain a for loop that inspects each input element. If an input element's value is I we increase C[i].  For each integer I = 0, 1,..., k, C[i] holds the number of input elements equal to i.  Lines 7–8 keep a running sum of the array C to determine how many input elements are less than or equal to I for each I = 0; 1,..., k.  **Termination :**  At the end: no remaining input  each number printed counts the number of times  As a result, the output has the same numbers as the input. |
| Three Strengths | 1. Counting sort generally performs faster than all comparison-based sorting algorithms. 2. Counting sort is easy to code 3. The Time complexity of this is O (n+k). |
| Three Weakness | 1. It is not suitable for large data sets 2. It is not suitable for string data sets 3. If the range of data is huge then it will consume very large amount of memory |
| Dry Run |  |

### Bucket Sort:

|  |  |
| --- | --- |
| Description | Bucket Sort which is also known as Bin Sort is a type of sorting algorithm in which we divide all the elements of the given array into a number of buckets. Then we proceed by sorting every bucket on individually basis. We can use either any sorting algorithm to sort bucket or can also implement by recursively calling the Bucket Sort algorithm. In bucket sort all the implementations are performed by comparisons, on that basis we can classify it as a comparison sort algorithm. The altogether time complexity analysis for bucket sort depends upon the type of algorithm we are using to sort the set of stacks.  In bucket sort we work by first initializing the arrays having empty buckets. Then we scatter the elements from the original array given in input and then put each element in its bucket. Then we proceed by checking all the buckets that are filled and sort them parallel. Then we gather the elements from all the buckets in specific order and rewrite them into the original array.  Bucket Sort has the concept that all the entries are taken from a uniform distribution. Bucket Sort is typically a fast algorithm as because we have used buckets in it instead of using an array where we have to manage the index values of arrays. This means that extra memory is required in bucket sort for buckets.  Bucket Sort is used when we have to manage a huge list of elements in a specific range and a list which is huge enough. Especially when we are sorting the arrays having float elements in its and they are given in a specific range that is like 1 to 5. At that time, we prefer to use bucket sort. Thus it is very efficient algorithm for such inputs. |
| Pseudo Code | Bucket-Sort(A)   1. Let B[0…n-1] be new array 2. n= A.length 3. **for** i = 0 **to** n-1 4. Make B[i] an empty list 5. **for** i =1 **to** n 6. Insert A[i] into B[nA[i]] 7. **for** i = 0 **to** n-1 8. Sort list B[i] with insertion sort 9. Concatenate the list B[0], B[1],….,B[n-1] together in order |
| Code in Python | def bucket\_Sort(A):  B= []  for i in range(len(array)):  B.append([])  for i in array:  B\_i = int(i \* 10)  B[B\_i].append(i)  for i in range(len(A)):  B[i] = sorted(B[i])  #insertion sort for buckets  k = 0  for i in range(len(A)):  for j in range(len(B[i])):  A[k] = B[i][j]  k = k + 1  return A |
| Time Complexity Analysis |  |
| Proof of Correctness | **Initialization:**  **The subarray A [1.... I-1] is being initialised, and this is the smallest element in the array.** |
| Three Strengths | 1. In bucket sort the elements of arrays are divided in their respective buckets. So, instead of sorting an array of large number of elements you can easily sort small elements of buckets 2. Bucket Sort is used when we have to manage a huge list of elements in a specific range and a list which is huge enough 3. It is a non-comparison sorting algorithm. So it has better time or efficiency than others. |
| Three Weakness | 1. Bucket sort cannot be applied for all types of data types. 2. If you do not able to distribute the data into their respective buckets than it may create problems. 3. The bucket sort performance depends upon the number of buckets formed, so large no of buckets lessen then performance. |
| Dry Run |  |

## Searching Algorithms:

Searching Algorithm are those algorithms in which we choose an element from the given data set and then find it from the entire data set that we have given. Searching algorithms also has a basis in our project. We have used multiple searching algorithms in our system to provide an environment for our user where they can search any entity by just typing and clicking the search button.

### Linear Search:

|  |  |
| --- | --- |
| Description | Linear search is a simplest search. In this we compare the item with each one until the match is found. This is a sequential search which is done all over the given data set that we have provided. |
| Pseudo Code | def linearSearch(A, n):    size=length(A)      for i =0 to size:          if (A[i] == n):              return i      return “NOT Found” |
| Python Code | def search(A, n, x):    size=len(A)      for i in range(0, size):          if (A[i] == x):              return i      return -1 |

### Binary Search:

|  |  |
| --- | --- |
| Description | The binary search formula is used with solely a sorted list of components. which means the binary search is employed only with a listing of parts that are already organized in an order. If the list is arranged in random order than the binary search cannot be applied on it. This search method starts comparison the search element with the center element within the list. If each are matched, then the result's "element found". Otherwise we can check that the element we have searched is it smaller or larger than the middle element of the array. If the search element is smaller, then we repeat a similar method for the left sub list of the center part. If the search element is larger, then we have a tendency to repeat the same process for the correct sub list of the middle element. |
| Pseudo Code | Procedure binary\_search(A , s , x)  lower = 1  upper = s  while x not found  if upper < lower  Break    set midPoint = lower+(upper-lower) / 2    if A[midPoint] < x  lower = midPoint + 1    if A[midPoint] > x  upper = midPoint - 1  if A[midPoint] = x  break |
| Python Code | Procedure binary\_search(A , x) :  lower = 0  upper= len(A) - 1  flag = False  while found == False and lower <= upper:  mid\_point = (lower + upper // 2  if A[mid\_point] == x:  print("number has been found.")  flag = True  elif A[mid\_point] < x:  lower = mid\_point + 1  else:  upper = mid\_point - 1  if flag == False:  print("Not found .") |

### Interpolation Search:

|  |  |
| --- | --- |
| Description | Interpolation search is Associate in Nursing algorithmic rule for finding out a key in an array that has been ordered by numerical prices allotted to the keys (key values). it absolutely was initial represented by W. W. Peterson in 1957. Interpolation search resembles the strategy by which individuals search a phonebook for a reputation (the key value by that the book' entries are ordered): in every step the algorithm calculates wherever within the remaining search area the wanted item may be, supported the key values at the bounds of the search space and also the value of the sought key, typically via a linear interpolation. The key price really found at this calculable position is then compared to the key value being sought. If it's not equal, then betting on the comparison, the remaining search area is reduced to the half before or when the estimated position. This methodology can solely work if calculations on the scale of variations between key values are wise |
| Pseudo Code | interPolationSearch()  strt =0  last = len(arr) -1  found = false  while found == false and start <=last    p = strt + (((last-start)/(A[last]-A[strt])) \* (num-A[strt]))  if A[p]==num    return p    if num > A[p]    strt = p+1    else    last = p-1  end while |
| Python Code | def interPolationSearch(A,num):  strt = 0  last = len(arr)-1  found = False  while strt <= last and found == False:  if strt == last:  if A[strt] == num:  return pos  return "not found"  p= strt + (((last-strt)//(A[last]-A[strt])) \* (num-arr[strt]))  if A[p] == num:  return pos  if A[p] < num:  strt = p + 1;  else:  last = p - 1;  return "Not Found" |

### Exponential Search:

|  |  |
| --- | --- |
| Description | The procedure of locating the preferred statistics from the set of gadgets saved within side the shape of factors with inside the pc reminiscence is known as looking.  Exponential search set of rules additionally known as doubling search, galloping search, Struzik search is a search set of rules created via way of means of Jon Bentley and Andrew Chi-Chih Yao in 1976 for looking sorted, unbounded/limitless lists. This mechanism is used to discover the variety in which the hunt key can also additionally present. If L and U are the higher and decrease certain of the list, then L and U each are the electricity of 2. For the ultimate section, the U is the ultimate role of  the list. For that reason, it's miles referred to as exponential.  After locating the precise variety, it makes use of the binary seek method to discover the precise vicinity of the hunt key. |
| Pseudo Code | ExponentialSearch(A):  s = size of the Array  if (s == 0)  return not found    bound = 1  while (bound < size && A[bound] < key)  bound = bound \* 2;    return binarySearch(A, key , bound/2) |
| Python Code | def ExponentialSearch(A, num):  s = len(A)  if (s == 0):  return "not found"    bound = 1  while (bound < s && A[bound] < num)  bound = bound \* 2  return binarySearch(A, num) |

## Searching Filters:

Exponential We can provide many kind of searching filters for users like:

1. In Integer Data type we will provide searching filter if the client will look in Year of Release he can particularly look in month of explicit year or date of explicit month.
2. In string we will give uncommon element that client can look exclusively by composing single letter set or numerous letter set.

## Multi-level Sorting:

Multi-level sorting will provide sorting on multiple columns For Example: If there are two movies starting with alphabet "A" it will check the second column and arrange it w.r.t director name according to ascending or descending order as per our choice.

# User Interphase Detail:

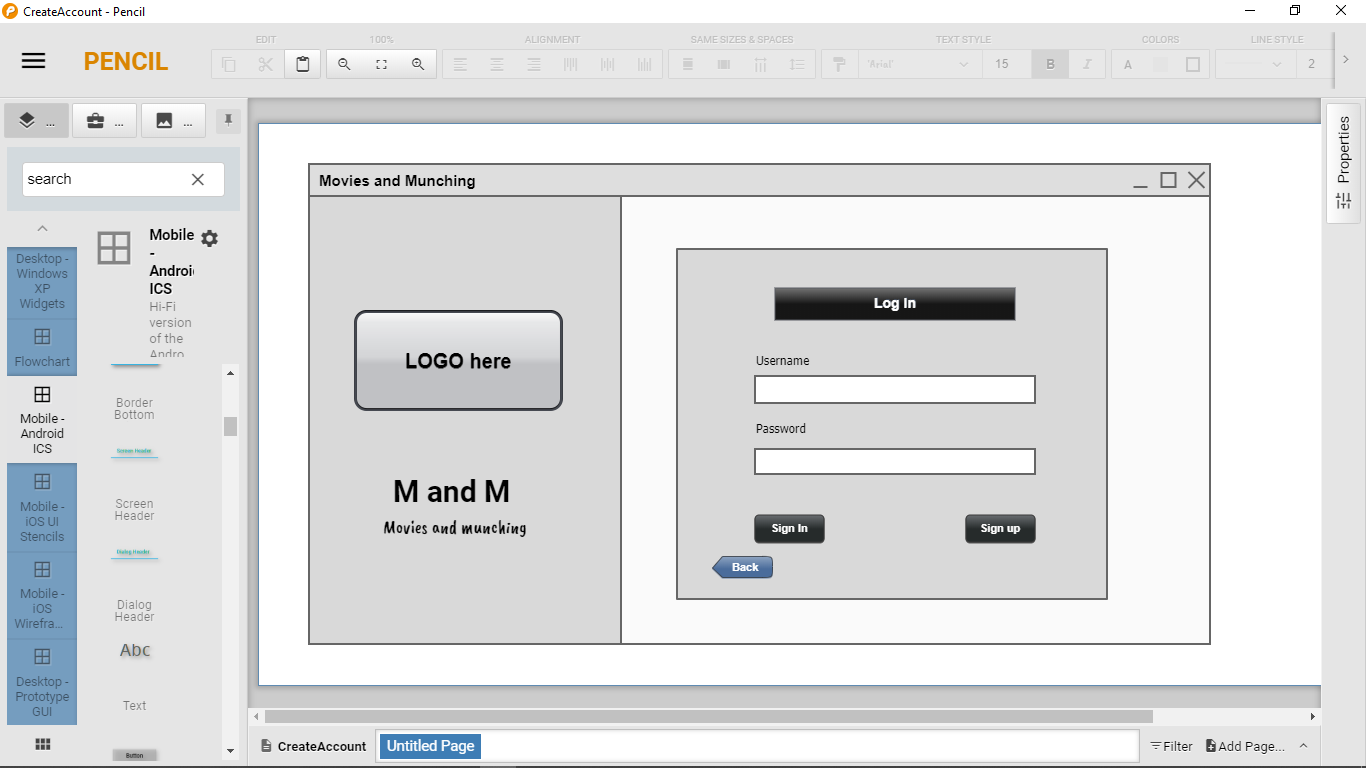
User Interphase also known as “UI” is an interaction between computer and a computer user. The user Interphase controls a software in background. The user is using the User Interphase which is connected to the background coding.

As the technology is growing the business on web, mobile and desktop applications has increases the demands of UI interphase. SO everyone is doing efforts on UI to increase their demands in the society. There are many type of User Interphases:

1. Graphical Interphase
2. Touch Interphase
3. Voice Interphase
4. Menu-driven Interphase
5. Command Line Interphase
6. Form Interphase

## Rough UI Look of Pencil Tool:

### Frontier:

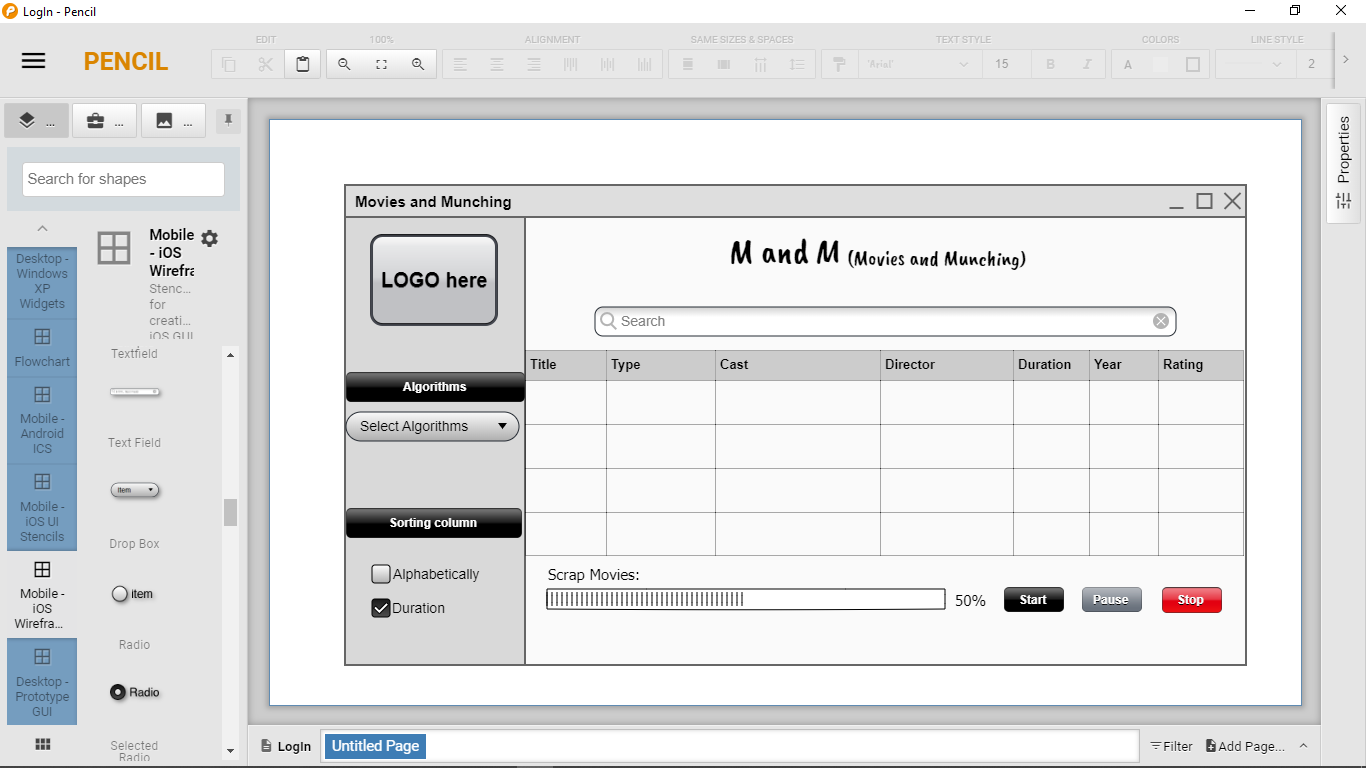
This is the first screen where the user has to press the enter button so that he can enters the main window of the project.

**Fig 5.1.1:** Screenshot of UI Design Pencil Tool.

|  |  |  |
| --- | --- | --- |
| **UI Component Name** | **Type of UI component** | **Purpose of UI Component/Other details** |
| UserName | TextBox | User will enter username here for login to main page |
| Password | PasswordBox | User will enter Password in this PasswordBox to login |
| Sign In | Button | This button will proceed user’s username and password |
| Sign Up | Button | If user have no username and password then this button will proceed the user to registration page. |
| Back | Button | End the program |

## Main Window:

This is the rough idea of main screen in Pencil Tool where the user has all the functionalities which we have provided in our project.



**Fig 5.1.2:** Screenshot of UI Design Pencil Tool

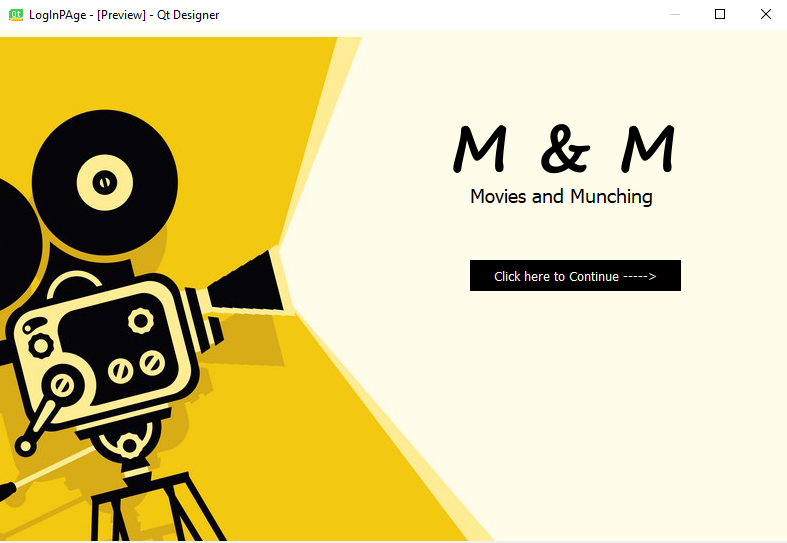
|  |  |  |
| --- | --- | --- |
| **UI Component Name** | **Type of UI component** | **Purpose of UI Component/Other details** |
| Search | SearchField | User can search movies from the table by entering the name of the movie |
| Select algorithms | ComboBox /  DropDownMenu | User can select any of the Algorithm provided in the drop down menu to sort the data of the each column |
| Sorting Column | Radio Button | User can sort the each column by selecting one of the provided radio button in Sorting Column section respectively |
| Scrap Movies | ProgressBar | Progress bar will show the amount of data that has been scraped from website at particular time |
| Start | Button | User can start scraping data from website by pressing this button |
| Pause | Button | User can pause scarping data from website by this button anytime |
| Stop | Button | User can stop scraping data by pressing this button anytime |
| Table | Table | All scarped movies/data will show in this table in sorted form |

## GUI Implementation:

This is the GUI implementation of our gui plan on PYQT designer and then is converted into the python code. This is according to all the requirements of our GUI system. We managed our GUI with various events to make it as much user friendly as we can.

### Frontier:

This is the frontier page of our project user interphase. We have provided the project name to user on this screen and then he can enter the main window by pressing the button that we have provided.



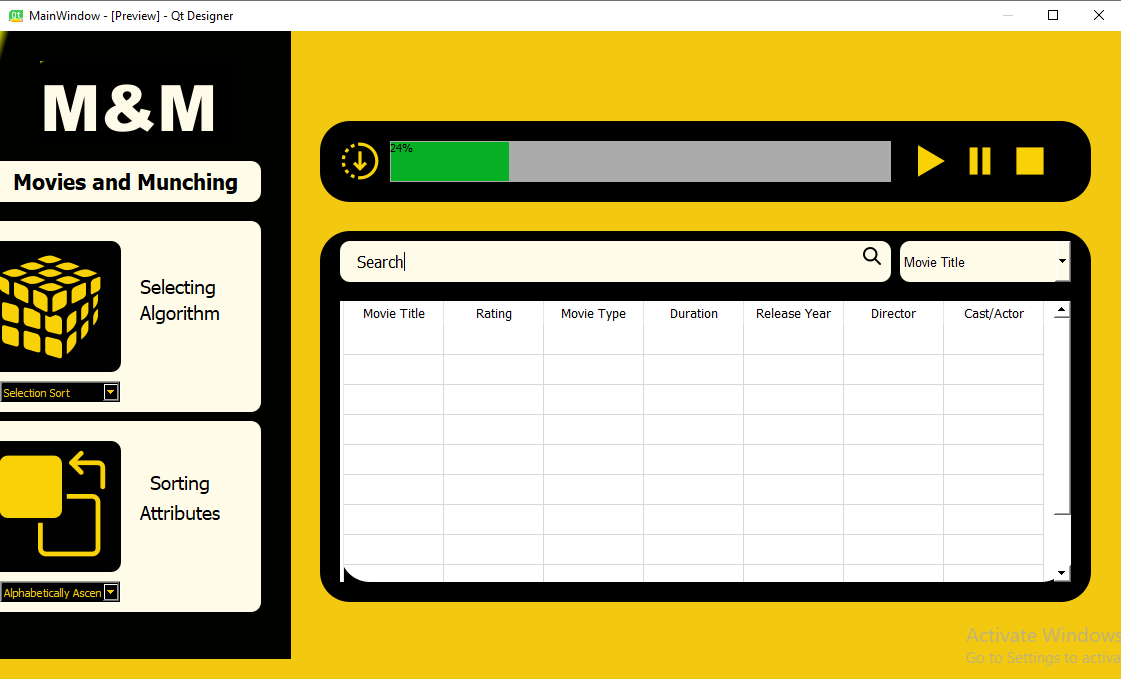
**Fig 5.2.1:** Screenshot of the frontier page of

M&M

|  |  |  |
| --- | --- | --- |
| **UI Component Name** | **Type of UI component** | **Purpose of UI Component/Other details** |
| pushButton | Button | User press this button to enter the main window of the program |

### Main Window:

This is the main window of our GUI implementation on PYQT5. We have converted it into in py code and from where we are implementing all of our functionalities.



**Fig 5.2.2:** Screenshot of the main window page of M&M

|  |  |  |
| --- | --- | --- |
| **UI Component Name** | **Type of UI component** | **Purpose of UI Component/Other details** |
| lineEdit | QLineEdit | User can search any entity of the movies from the table by entering the name of that entity in this line edit. |
| comboBox\_3 | ComboBox /  DropDownMenu | Here user will select the entity of the movie from where he want to search . |
| label\_5 | Label | This label contains a search icon. On clicking the search begins |
| comboBox | ComboBox /  DropDownMenu | User can select any of the Algorithm provided in the drop down menu to sort the data of the each column. |
| label\_5 | Label | This label contains a algorithm icon. On clicking it the sorting begins. |
| comboBox\_2 | ComboBox /  DropDownMenu | User can sort the each column by selecting one of the provided options in Sorting Column section respectively |
| progressBar | ProgressBar | Progress bar will show the amount of data that has been to the table at particular time. |
| label\_6 | Label | This label contains a start icon. On clicking it the data loading starts begins. |
| label\_12 | Label | This label contains a pause icon. On clicking it the data loading paused. |
| label\_11 | Label | This label contains a stop icon. On clicking it the data loading atoped. |
| tableWidget | Table Widget | All scarped movies/data will show in this table . |

# Integration:

Project Integration is the coordination of all the aspects of the project and how well do you managed them. This includes many aspects like coordination of project members, managing conflict among themselves and various aspects of the projects There they assess all the resources and combine them together in a meaningful system. How to evaluate the situation and making informed decisions is an important part of integration. Integration is related with the whole management of the project. You have to keep an account of all the aspects and elements of the project to get a best result.

## Scrapping:

As our project is a scrapping based project so we get a start with the scrapping procedure. Indeed, we have to decide a website according to our requirements. We have a task that we have to scrap at least seven entities of the data we are scrapping. Thus our first hurdle is to find a website from where we can fulfill our requirement.

Data that provided reliable information for occupations had to be collected in a worksheet. Progressive phases of job aggregator, workforce organization and recruitment must create correct stock of materials according to labor market interests.We were allocated to collecting such data, which should be consistent with the applicant's work plan and needs. So after getting through many websites we have selected the following website for scrapping:

* **IMDB**

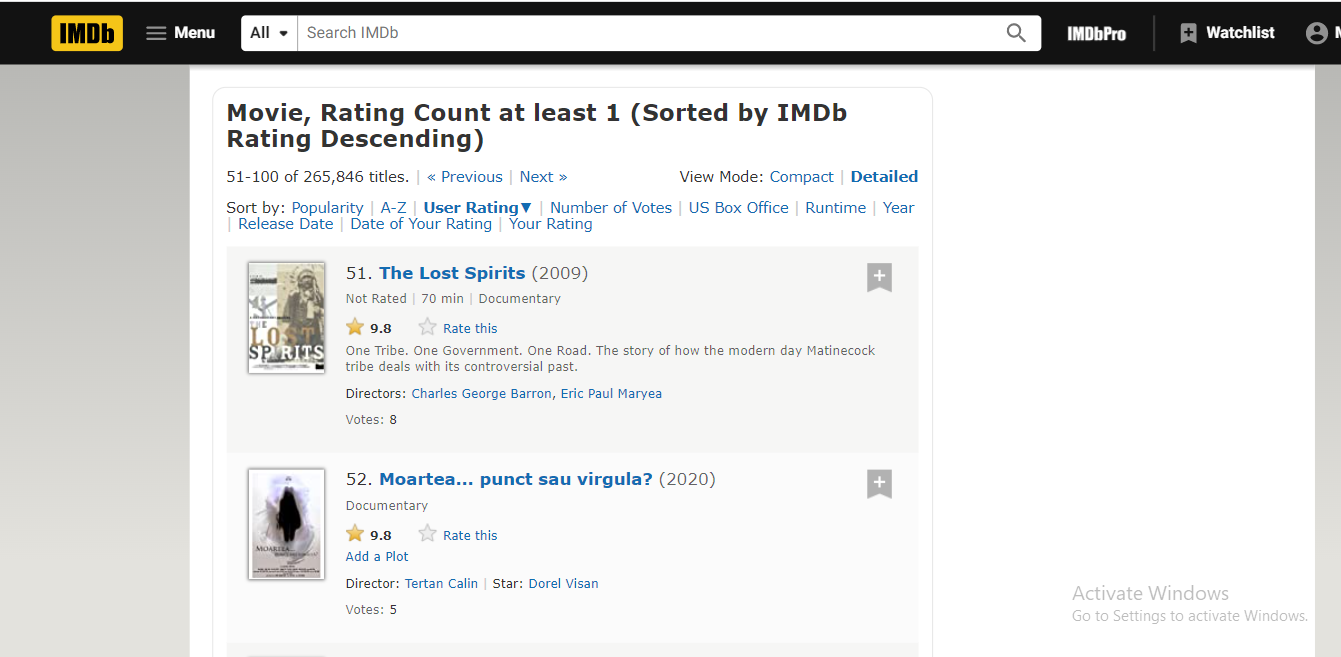
Below are the snapshots and link of the website from where data is scrapped.

### IMDB:

IMDb **(Internet** Movie **Database)** is an online database of information related to **movies,** television programs. IMDB is the popular website for movies and authoritative source of all kind of movies and television programs. Here we have found all the requirements that we were expecting. According to an analysis in June 2021 the IMDB website have around 8 million of titles and 10.4 million personalities. Thus it is one of the nest websites for movies. Thus our purpose of scrapping 1 million of data would also be fulfilled from this website.

**Website Link:**

<https://www.imdb.com/search/title/?title_type=movie&num_votes=1,&sort=user_rating,desc&start=51&ref_=adv_nxt>



**Fig 6.1.1:** Snapshot of IMDB website

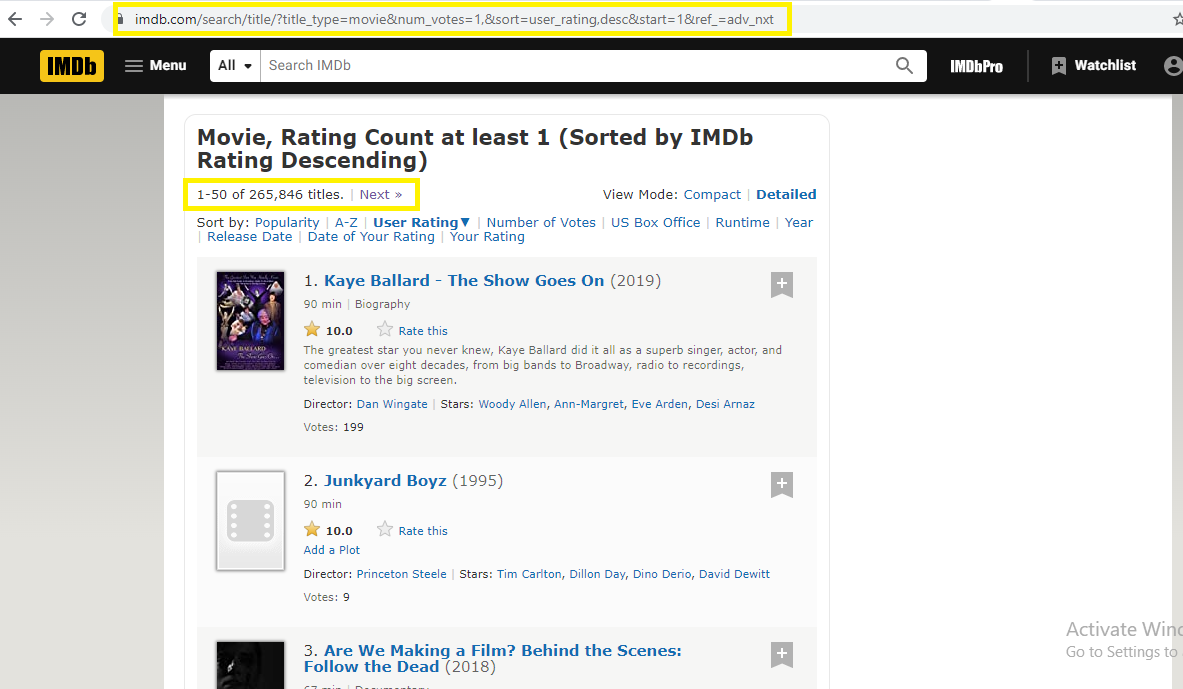
Following attributes, we scrap from this website:

* + Movie name
  + Director name
  + Movie Type
  + Movie Year
  + Movie Actor
  + Movie Rating
  + Duration

### Difficulties:

First we begin to scrap the data by using BS4(Beautiful soap). We make a generic change in hyper link of website and then organized it to manages by iterating through loops. As there are about 50 movies in one page. Then by observing the link of website we note that by moving the next page there is an increment of 50 in the link.

**Initially:**

[https://www.imdb.com/search/title/?title\_type=movie&num\_votes=1,&sort=user\_rating,**desc&start=1**&ref\_=adv\_nxt](https://www.imdb.com/search/title/?title_type=movie&num_votes=1,&sort=user_rating,desc&start=1&ref_=adv_nxt)

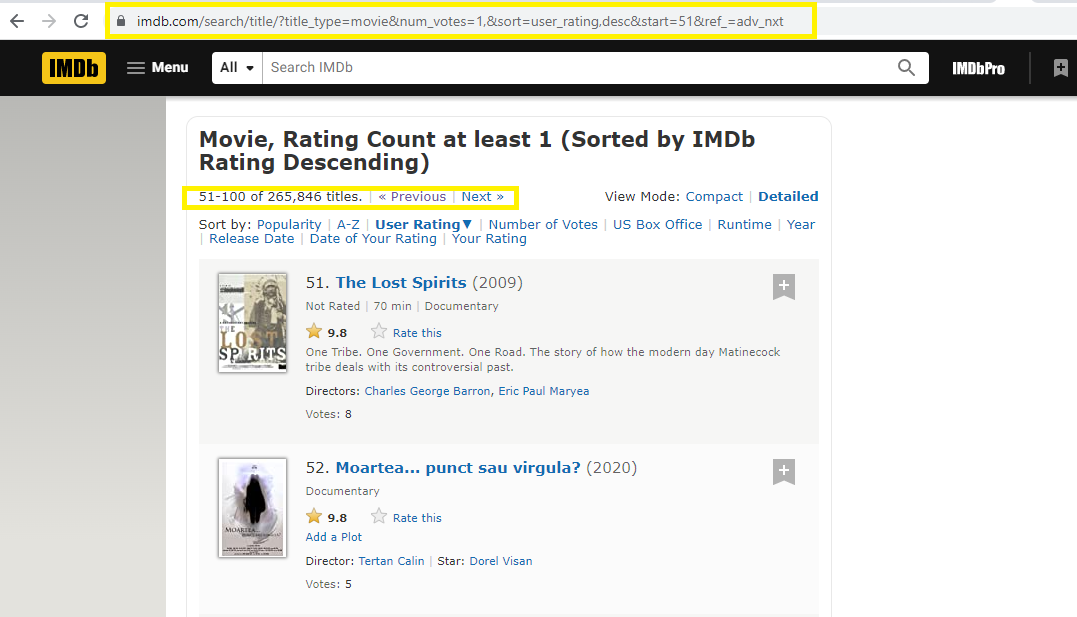
**Fig 6.1.2(a):** Snapshot showing the page load for the link given above

We can encounter the initially the link contains the (**desc&start=1).** Then by moving the next page there in an increase of 50.

**After:**

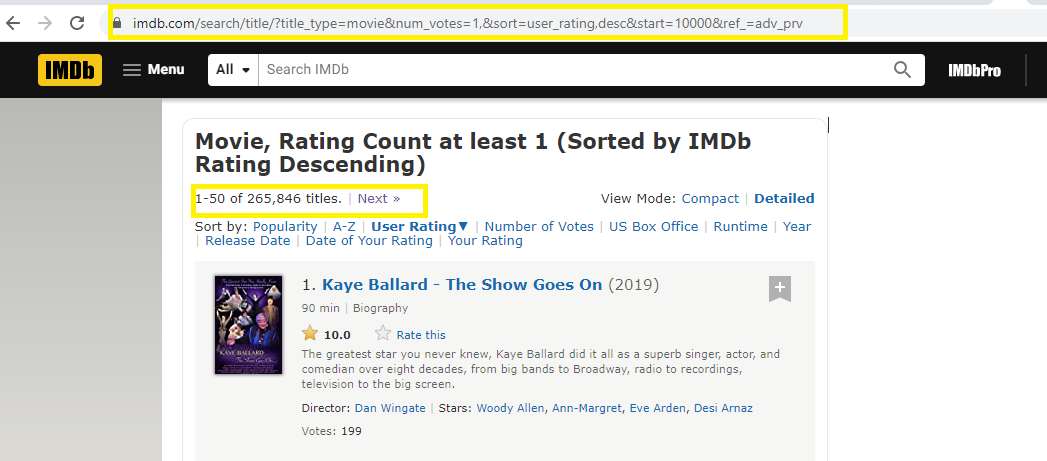
[https://www.imdb.com/search/title/?title\_type=movie&num\_votes=1,&sort=user\_rating,**desc&start=51**&ref\_=adv\_nxt](https://www.imdb.com/search/title/?title_type=movie&num_votes=1,&sort=user_rating,desc&start=51&ref_=adv_nxt)

Now for next page the factor **(desc&start=51)** has increased by 50 thus initially we use the for loop to make an increment of 50 to move to next page and scrap data.



**Fig 6.1.2(b):** Snapshot showing the page load for the link given above

**Problem:**

When we reached for the increment in link above 10000 then the link shows abnormality. There he continues to load the first page again and again.

**Fig 6.1.2(c):** Snapshot showing the page load for the link for (**desc&start=10000).**

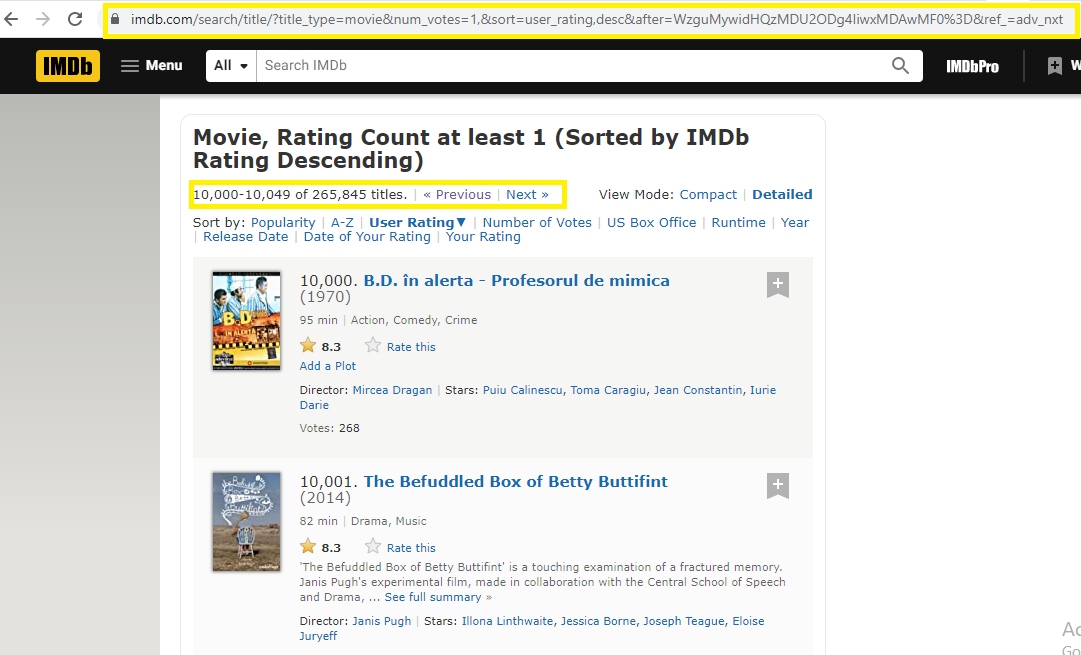
**Link:**

[https://www.imdb.com/search/title/?title\_type=movie&num\_votes=1,&sort=user\_rating,**desc&start=10000**&ref\_=adv\_prv](https://www.imdb.com/search/title/?title_type=movie&num_votes=1,&sort=user_rating,desc&start=10000&ref_=adv_prv)

Thus the same page loads again and again for the factor **(desc&start=10000).** So there is an repetition of data again and again after this link.

**Actual Link:**

[https://www.imdb.com/search/title/?title\_type=movie&num\_votes=1,&sort=user\_rating,**desc&after=WzguMywidHQzMDU2ODg4IiwxMDAwMF0%3D**&ref\_=adv\_nxt](https://www.imdb.com/search/title/?title_type=movie&num_votes=1,&sort=user_rating,desc&after=WzguMywidHQzMDU2ODg4IiwxMDAwMF0%3D&ref_=adv_nxt)

****

**Fig 6.1.2(d):** Snapshot showing the page load for the above link

Thus the factor **(desc&after=WzguMywidHQzMDU2ODg4IiwxMDAwMF0%3D)** become abnormal at this point and it’s impossible to make a generic link ahead from this.

### Solution:

In order to overcome this problem, we transfer from BS4(Beautiful Soap) to web driver to get all the data from the web page. Web Driver is an automatic framework which works by controlling the web pages through open APIs. Its works by reading the given commands then deliver them to the web page and then perform the given command. Thus when we implement the web driver then we were able to control the entire web page.

Thus we proceed by just giving the first page link and then clicking on the next button on web page by using the web driver and moves to next page.

We use the following command to click on the next button and to move next page:

**driver.find\_element(By.XPATH, "(//a[@class='lister-page-next next-page'])").click()**

This thing worked for us and we overcame our above problem as described.



**Fig 6.1.3:** Snapshot of the next which we clicked to move next page

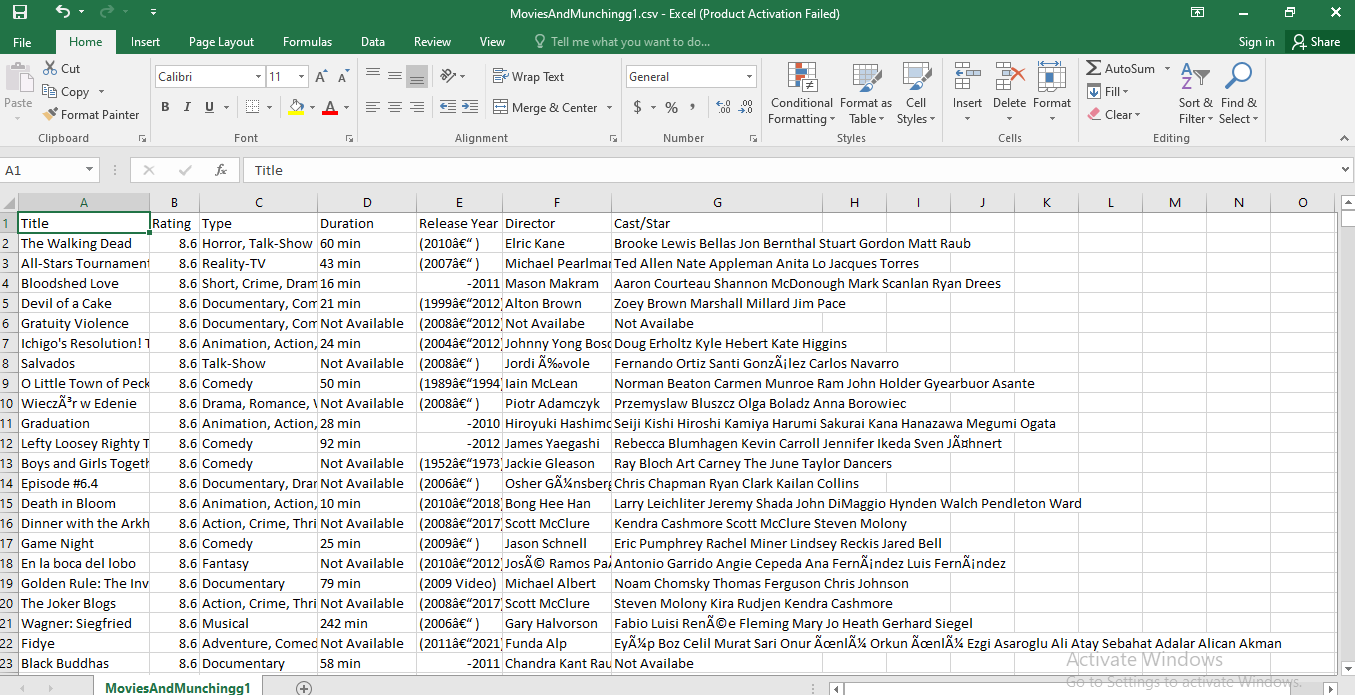
## Data Storage:

After the data is scrapped now we have to store a lot of data in some storage files. So we can use the given data in near future when we want to use it. Data is major source for any project. Thus we store it in storage files to keep it safe.

### Excel:

As we are advised to scrap 1 million of data for our project. So we are able to scrap all of the 1 million of data of movies. Then we save the whole data in the CSV file to store it on safe place and to make it useable in the future. The data in CSV is save as followed:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Title | Rating | Type | Duration | Release Year | Director | Cast/Star |



**Fig 6.2.1:** Snapshot showing data save in CSV

## UI Integration:

User Interphase also known as “UI” is an interaction between computer and a computer user. The user Interphase controls a software in background. The user is using the User Interphase which is connected to the background coding.

As the technology is growing the business on web, mobile and desktop applications has increases the demands of UI interphase. SO everyone is doing efforts on UI to increase their demands in the society. There are many type of User Interphases:

1. Graphical Interphase
2. Touch Interphase
3. Voice Interphase
4. Menu-driven Interphase
5. Command Line Interphase
6. Form Interphase

### Requirement:

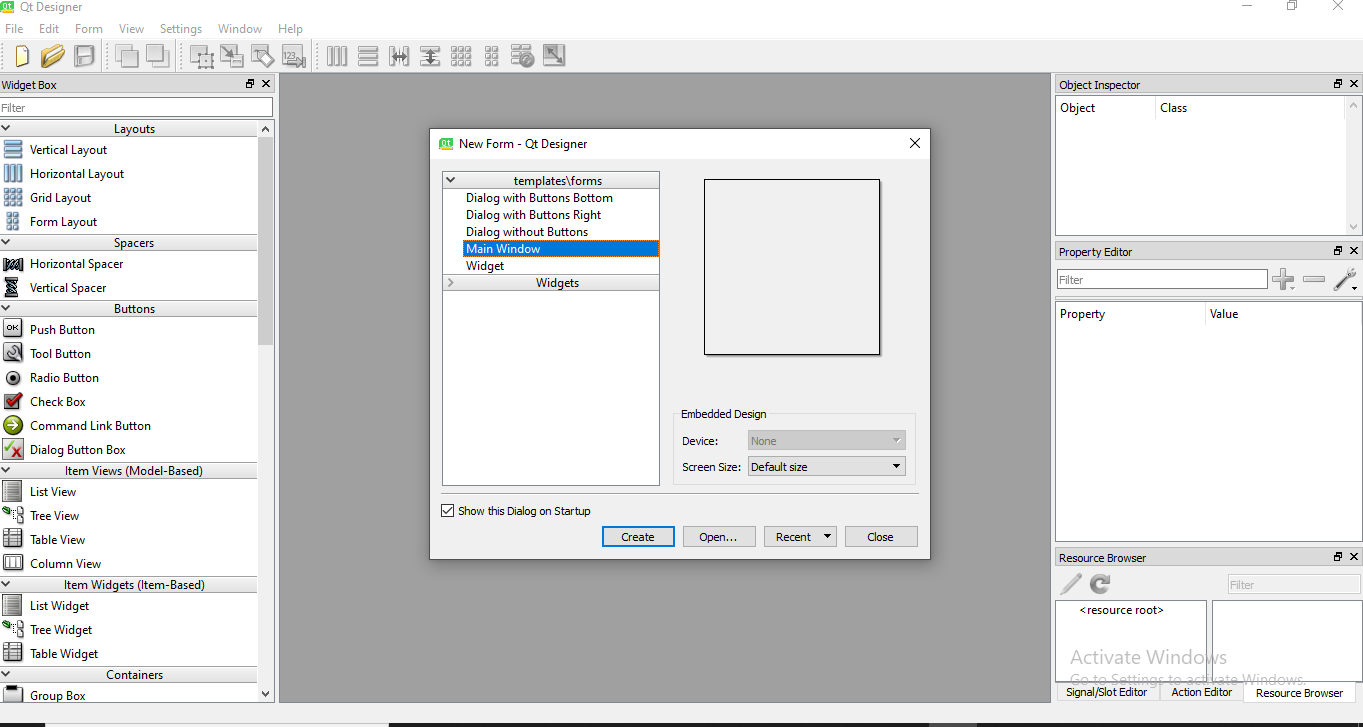
As we are asked to make a GUI for our project. We are asked to make a main screen which contains all the functionalities of our project. Discussing some of them is that we have to show all the data in the table, given the option to sort the data by using any sorting algorithms. Plus, we have to create an environment where the user enters the text that he wanted to find from any of the entire columns.

We are also advised to show a progress bar where the percentage of load data has been showed. There should be an option to pause, start and stop the loading of data in the table.

We are also advised to use the particular IDEs for creating the GUI. We are advised to work on pyqt5 designer.

### Pyqt Designer:

PYQT Designer is a tool where we build GUIs for our desktops. With this designer you don’t have to write the codes but you can just only drag and drop the icons according to your needs. Then you can modify it by using the style sheets of every widget.

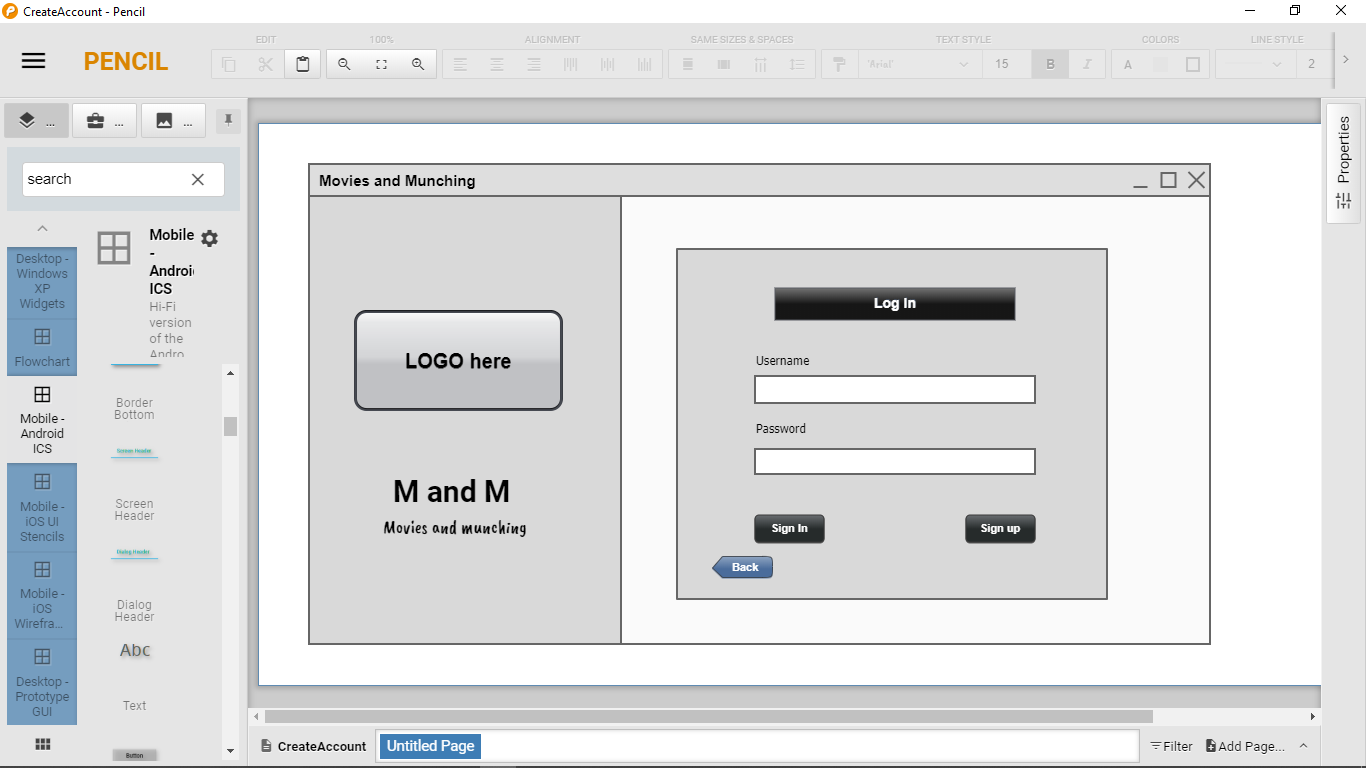
The widgets use in this designer are already customized. It doesn’t produce any code in any programming language. Rather, it creates an UI. File which we can convert to any language.

**Fig 6.3.2:** QT Designer IDE

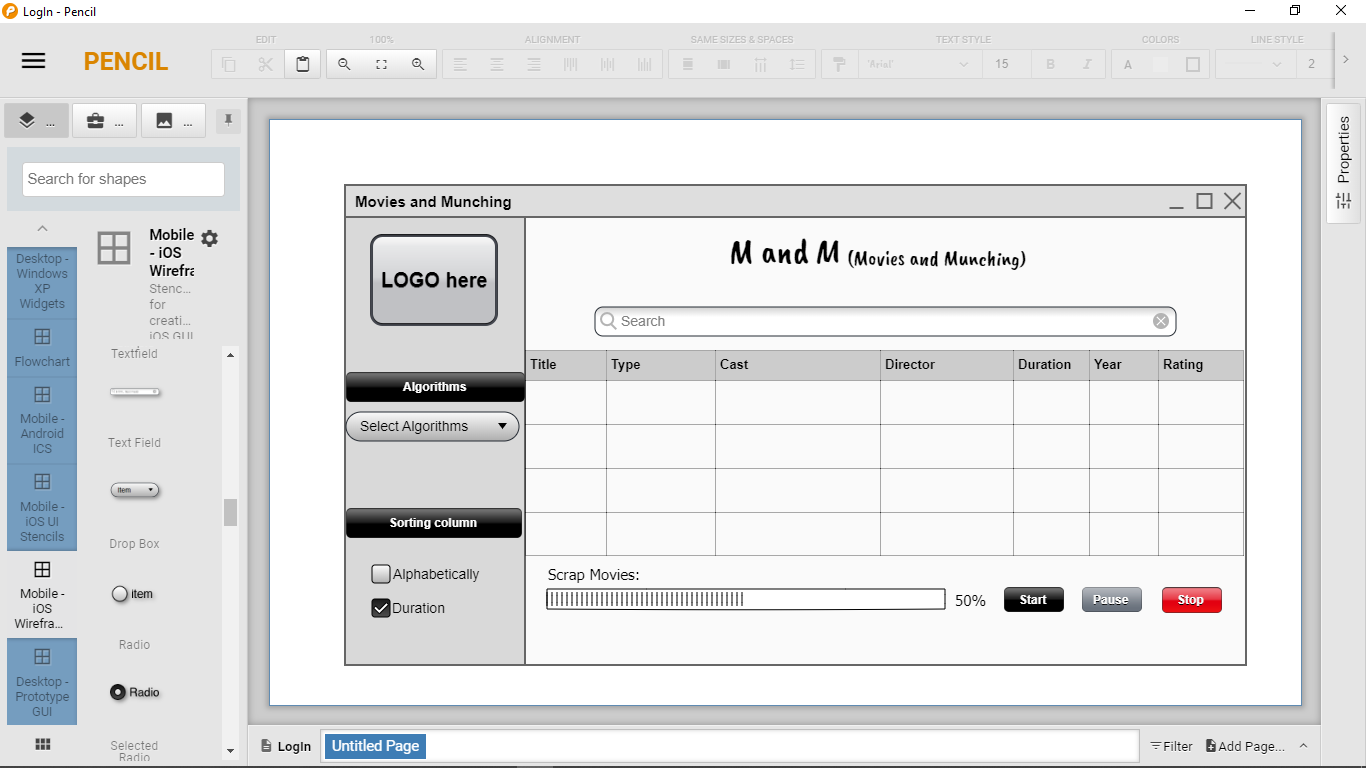
### Rough Look:

It is a good practice to create an idea or rough look before implementing it. Thus in order to produce a good GUI interphase first we implement our idea through pencil tool. This gives us a domain that in which direction we are moving. It redirects our path towards a software where we can have built much better then it.

**Login:**



**Main Screen:**

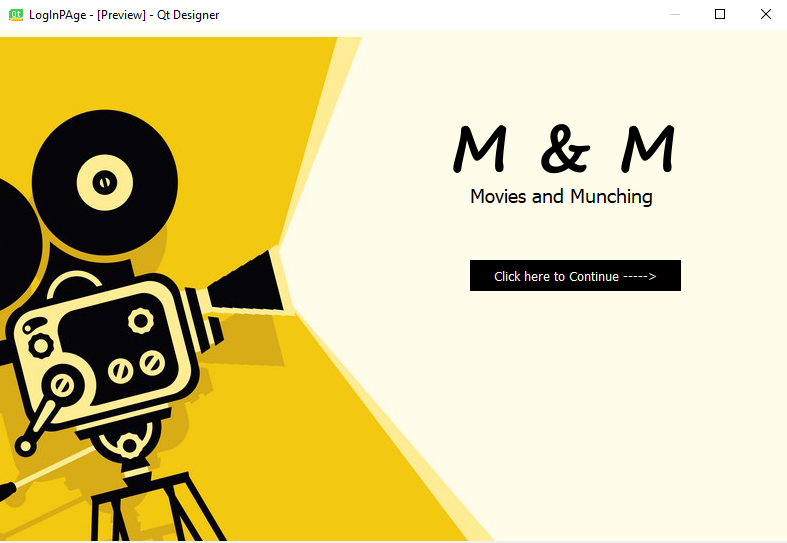


**Fig 6.3.3: Pencil Tool Designs**

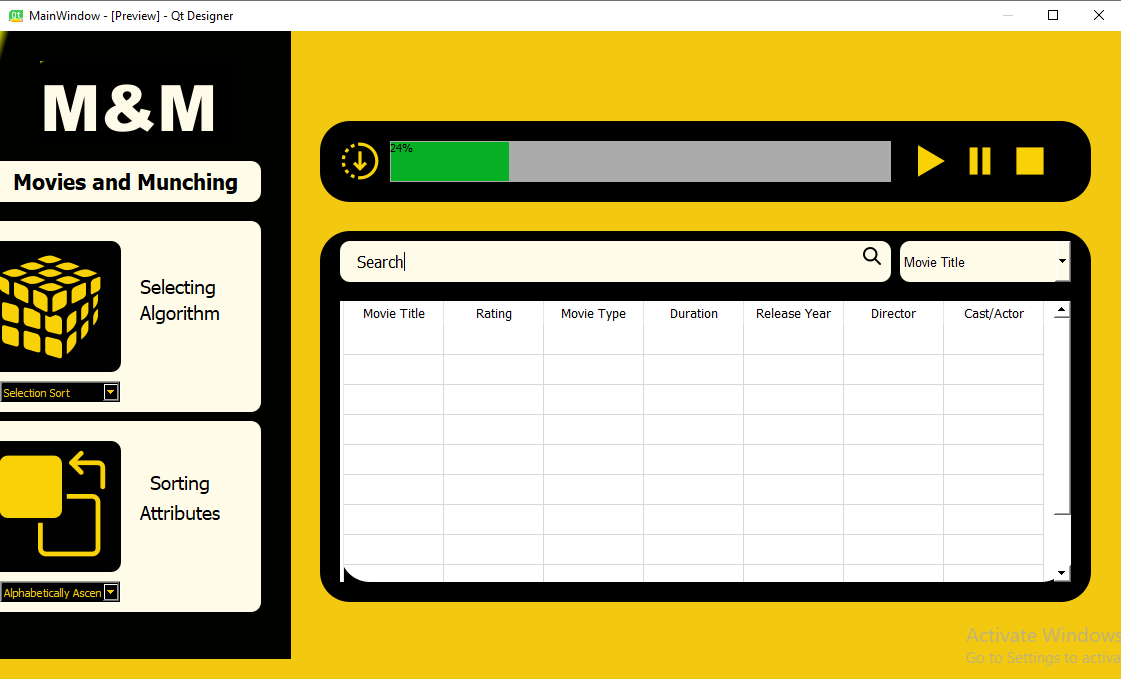
### Designer Implementation:

PYQT designer implementation after fully collaboration is as follows:

**Login:**



**Main Screen:**



**Fig 6.3.4(a):** GUI Designs

Thus you can see the major difference between the implementations and idea of an UI implementation. In order to elaborate let us view all the widgets in detail to study GUI:

* **QTable Widget:**

QTable Widget is used to show all the data of the GUI.



**Fig 6.3.4(b):** Qtable Widget

* **Search Bar:**

Search bar is made of the combination of Line Edit and labels along with a combo box.



**Fig 6.3.4(c):** Search Barr

* **Progress Bar:**

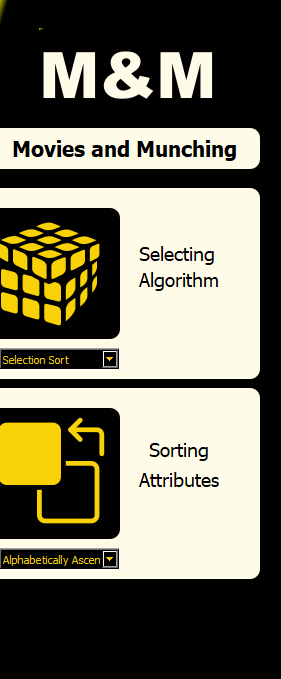
Progress Bar shows the percentage of data load in the table.



**Fig 6.3.4(d):** Progress Bar

* **Selection Menu:**

Through this menu the user can select that weather which type of sorting he wants to apply. We have added many labels and combo box to get the menu functional. On clicking the icons, the task proceeds.



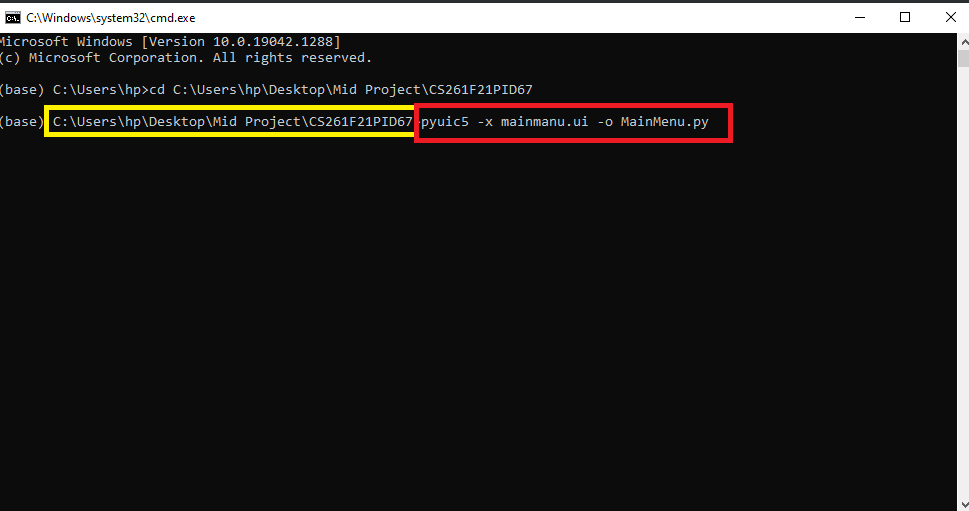
**Fig 6.3.4(e):** Selection Menu

## Backend Execution:

Once we are done with our front end GUI after that that we move towards the execution of our code or project. Now accessing the GUI, we can implement different methods to carry out the execution towards the final process.

### Conversion of files:

As I have described before that the QT designer do not produce any kind of programming code file. Thus we have to convert .ui file to .py files. For this we have to go to command prompt through anaconda and launch it from there. Then from cmd prompt you have to go into the directory where .ui file is placed. Then after giving some commands you can convert them as follows:

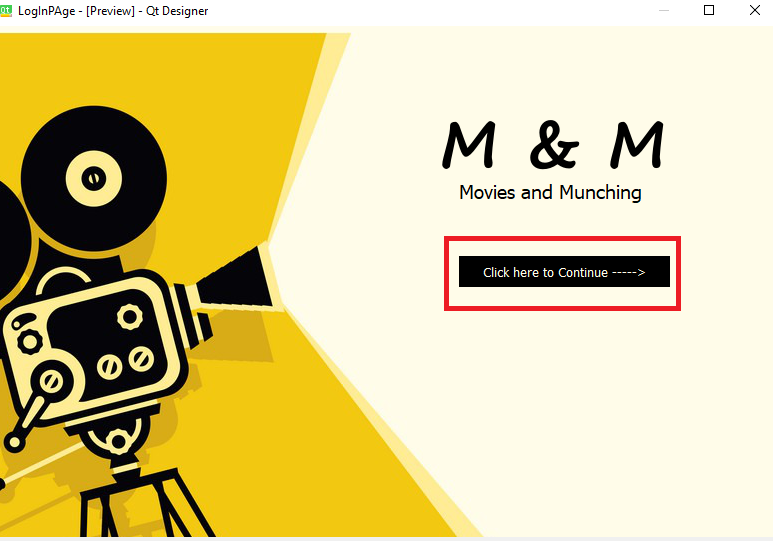


**Fig 6.4.1:** Snapshot of converting .ui to .py files on cmd

* The yellow line in the above figure is showing the path of the directory.
* The Red line is showing the command for converting the .ui to .py file.

### Log in:

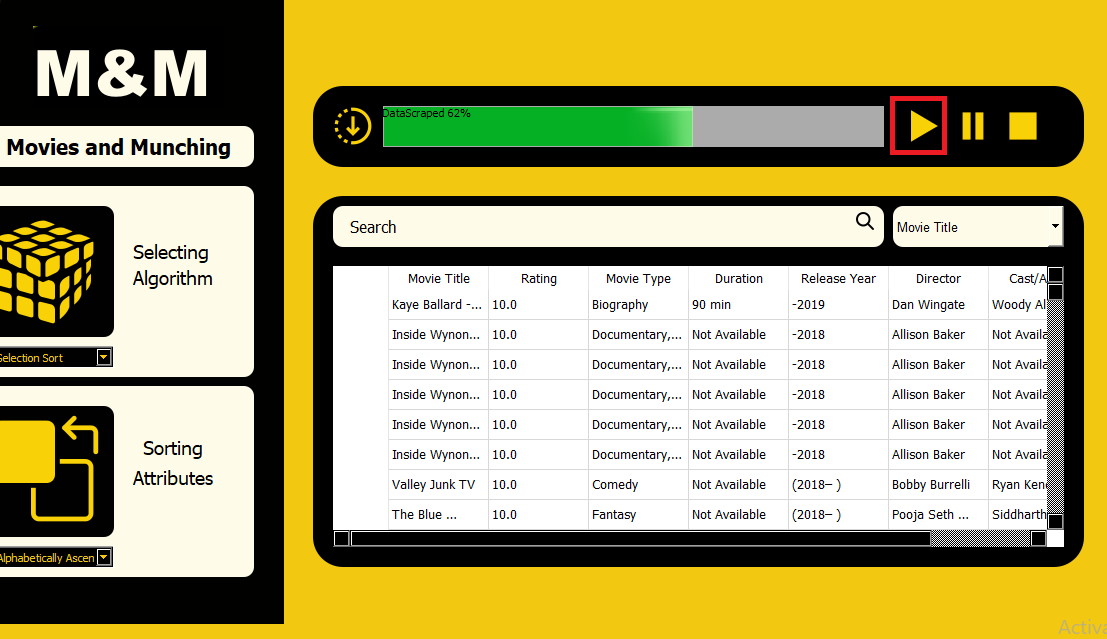
We have chosen a login screen just to carry the attention of the user. From there just by pressing the button the user can enter the main menu.



**Fig 6.4.2:** Snapshot of button to enter main window

### Data in Table:

QTable widget are the standards table that we are using in his project to show all the data. On clicking the play button, the data starts to upload on the table as follows:



**Fig 6.4.3:** QTable Widget