Criterion B: Design

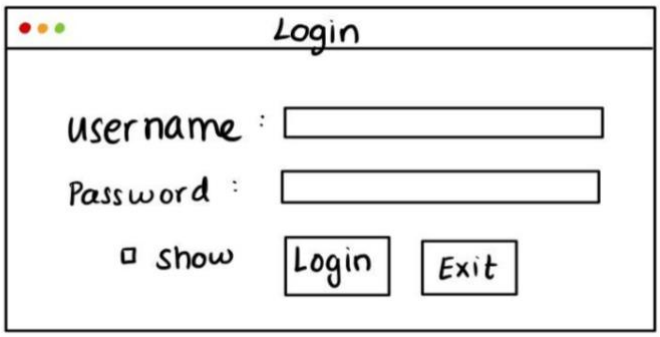
Interface Design (Front-End)

The desktop application's user interface design was conceptualized taking into account the requirements of the client as shown below:

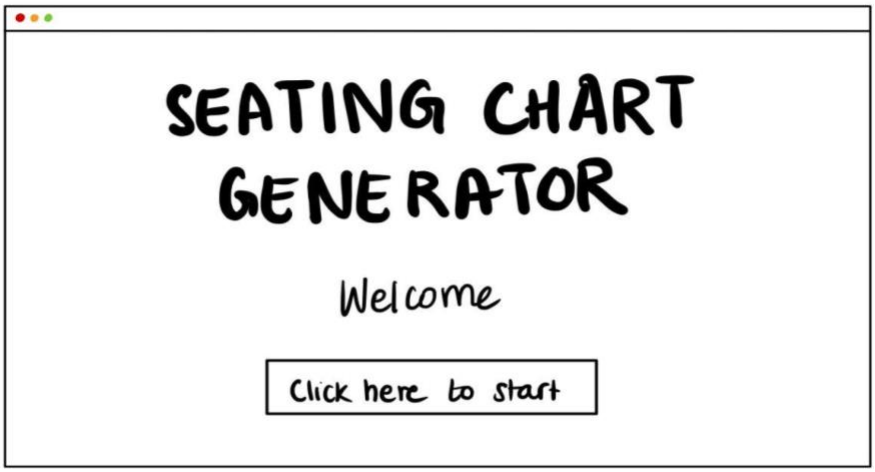
- Log-in screen with multiple users for different teachers
- 'Start' button to redirect the user to student data
- Viewing class data in an organized tabular manner.
- Buttons to update class data including 'Edit', 'Delete', 'Input New'
- Button to Calculate Weighted Scores of each student
- Window to choose group size followed by Button to Generate Seating Charts.

The following preliminary screens were designed for the user interface:

Screen 1: Login



Screen 2: Welcome Screen



Screen 3: Main Home Screen

ID	Name	Completed Assignments	Test 1	Test 2	Attendance	Behaviour Rating
1	Sam	8	19	20	93	9
2	John	9	16	18	94	10
3	Julie	7	13	11	86	8
4	Jane	10	15	13	89	7
5	katie	6	17	18	86	8
6	Joe	3	9	11	87	6
7	Rachel	5	12	15	72	5

Student ID | Student Name | Compl. Assignments | Test 1 | Test 2 | Attendance % | Behaviour Rating

Update Record | Delete Record | Refresh weighted scores

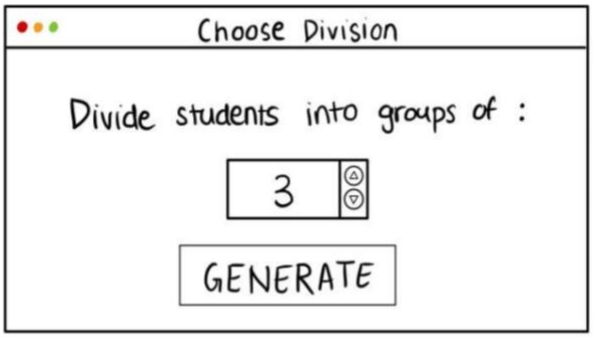
Save Update | Input Record | Generate Seating Chart

Screen 4: Display weighted scores

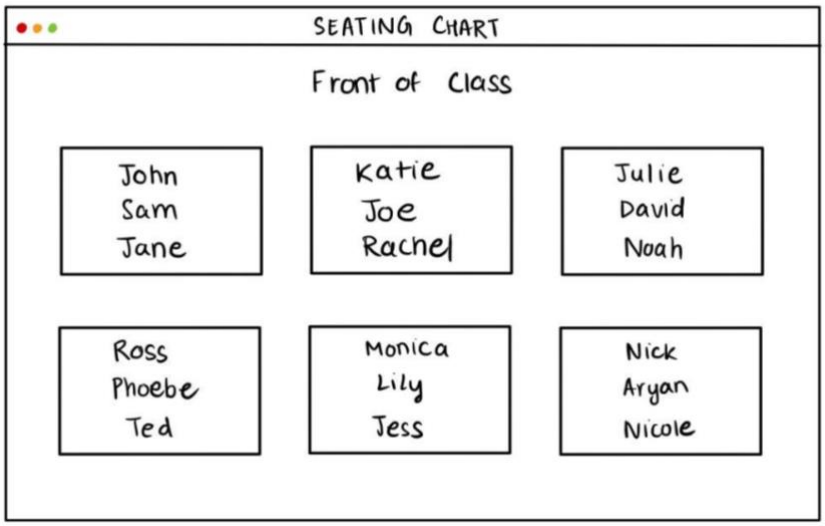
Student Weighted Records

Student ID	Name	Weighted Score
1	Sam	0.89
2	John	0.78
3	Julie	0.92
4	Jane	0.73
5	Katie	0.62
6	Joe	0.84
7	Rachel	0.59

Screen 5: Choosing group size



Screen 6: Visual display of seating chart



Tkinter

Each screen will be created using Tkinter – a Graphical User Interface library for Python which allows for the programmatic use of widgets. Widgets are objects, instances of classes representing buttons, drop down lists, user input entry, labels, frames, tables, etc.

Visuals

The client requires the seating chart to be displayed in an understandable visual manner for students and teachers. Thus, the seating arrangements will be displayed using Tk Widgets, allowing for ease of comprehension from users.

MySQL

In order to store and refresh current student data for different PYP teachers, the relational database management system (RDBMS) MySQL will be used. MySQL can be used with the programming language Python with the help of the driver 'MySQL connector'.

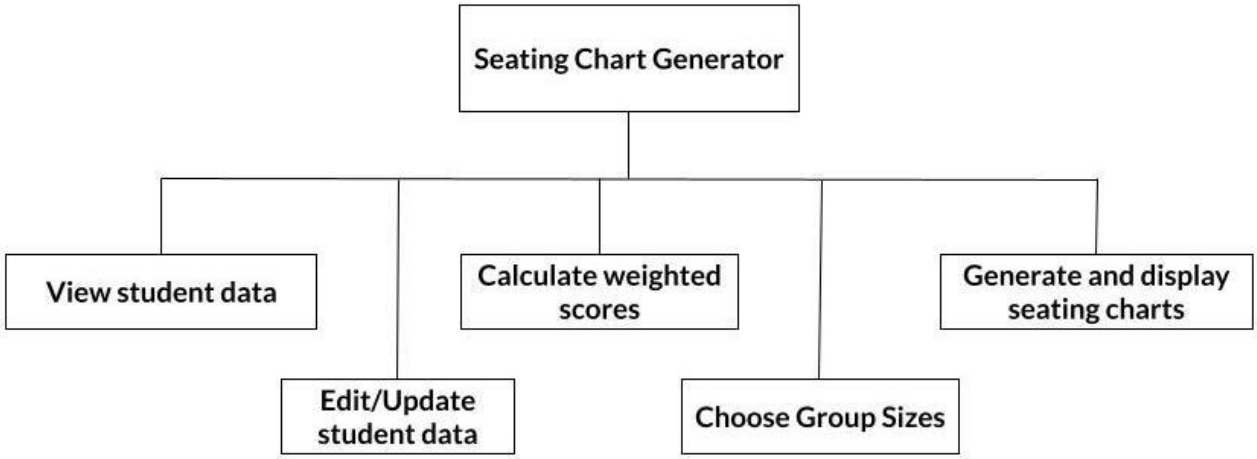
Other Libaries

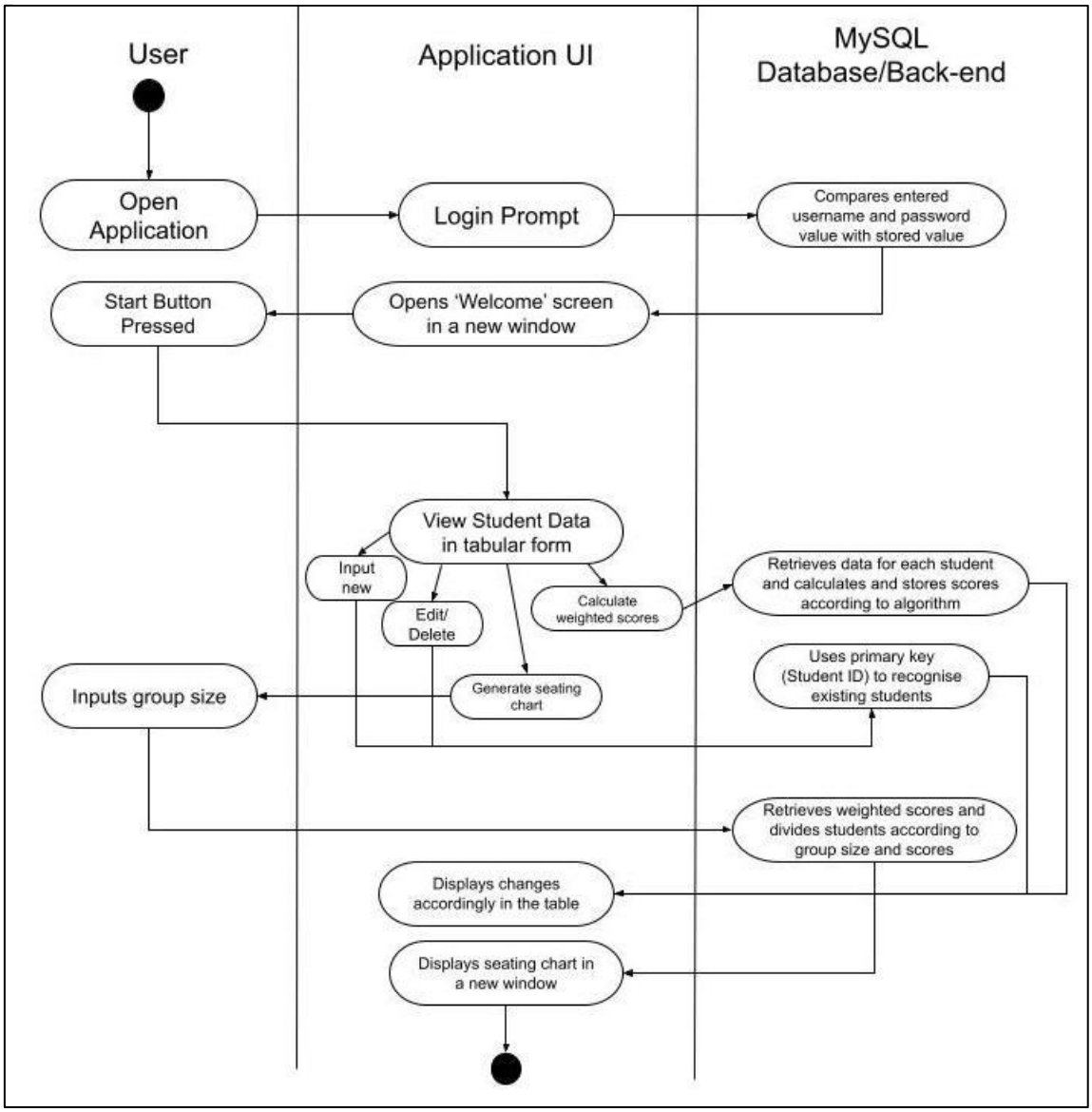
1. PyAutoGUI
   This library is used to automate mouse cursor moves, clicks, and keyboard presses. Since the client requires the seating chart to be exported/saved, this library will help automate the screenshot process.

2. Python Imaging Library
   This library helps in displaying images in the GUI.

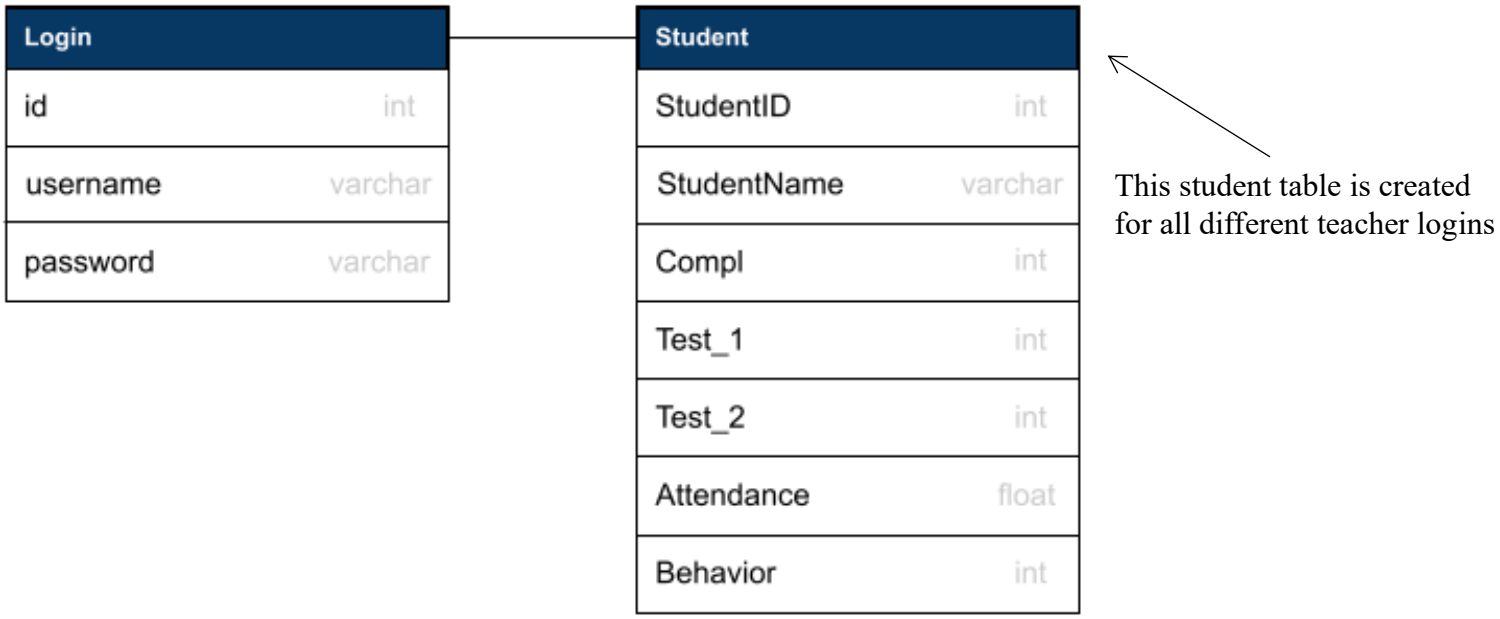
3. Math
   This library provides access to common math functions.

The project architecture may be represented as follows:





MySQL Database Structure:



Extensibility and Scalability

Even though the client specified that they only require a maximum number of 30 students in one class, because my product has a database integrated, it is extensible and can accommodate for more than 30 students.

Algorithmic Design

Calculating Weighted Scores

```
# Assignments Completed = 15%
# Attendance = 20%
# Best Two Tests = 35%
# Monthly Class Behaviour Rating = 30%

# Assignments completed

create list ‘studentData’ to hold number of completed assignments for each student
# Extracted from MySQL database

create list to hold percentage of completed assignments for each student

loop x from 0 to length(studentData)
      assignmentsCompleted.insert(studentData[x] / 15)
      # 15 assignments are given in total to each student monthly as informed by
      the client
end loop

# Attendance
```

```
create list ‘studentData’ to hold each student’s percentage attendance
# Extracted from MySQL database

create list to hold percentage attendance for each student

loop x from 0 to length(studentData)
      attendance.insert(studentData[x] / 100)
      # The client will directly input percentage attendance for each student,
      hence dividing by 100 simply gives a decimal value
end loop

# Behaviour rating

create list ‘studentData’ to hold each student’s behaviour rating
# Extracted from MySQL database

create list to hold behaviour rating out of 10 for each student

loop x from 0 to length(studentData)
      behaviour.insert(studentData[x] / 10)
end loop

# Best 2 tests

# Extracted from MySQL database
create list ‘studentData’ to hold each student’s score for test 1
create list ‘studentData1’ to hold each student’s score for test 2

create list to hold average test score for each student

loop x from 0 to length(studentData)
      avg = (studentData[x] + studentData1[x] / 2)
      avg = (avg/20) # As informed by client that tests are scored out of 20
      testScore.insert(avg)
end loop

# Calculate the weighted score out of 1

create list to hold weighted scores for each student

loop i from 0 to length(behaviour)
      finalScore = assignmentsCompleted[i]*0.15 + attendance[i]*0.2 +
      testScore[i]*0.35 + behaviour[i]*0.3

      finalScore = round(finalScore, 2) # where 2 represents the number of decimal
      places to round finalScore to

      weightedScores.insert(finalScore)
end loop

# Names
```

```
# Extract names from MySQL Database in list 'names'

create list 'scores_names' to hold weighted scores and names in a tuple

loop x from 0 length(names)
      scores_names.append(weighted scores[x], names[x])
end loop
```

Find the number of rows and columns

```
# Function to find number of rows and columns needed for a seating chart according
to number of students using the existing math library in Python

Func find_rows_cols(num): # num represents number of students
      cr = math.sqrt(num)

      # Checking if the squareroot of num is an integer
      if isinstance(cr, int) = True:
            col = cr
            row = cr
            return col, row
      else:
            cr = math.ceil(cr)
            col = cr
            row = cr
            return col, row
```

Print Tkinter Buttons in a grid format

```
# Function to print Tkinter buttons in grid format to display seating chart
Func print_chart(count, len, col, row):
      loop while count ≠ (len-1):
           loop x from 2 to col:
              loop y from 2 to row:
                   print Tkinter buttons(row=x, column=y)
                    count = count + 1
               end loop
            end loop
      end loop


Func generate():
      groupSize = int(entry.get()) # User input from Tkinter entry box
      len = length(scores_names)
      col, row = find_rows_cols(len)
      # col, row are used to place Tkinter buttons in a grid format

      if groupSize = 2:
            create list 'pairs' to hold names of students in pairs
            if len % 2 = 0:
```

```
            loop x from 0 to (len // 2):
                  # grouping the highest score with the lowest score
                  pairs.insert((scores_names[x], scores_names[len - 1]))
                  len = len – 1
            end loop
            count = 0
            run func print_chart(count, len, col, row)

      else: # odd number of students
            temp = scores_names[(len-1)//2]
            scores_names.pop[(len-1)//2]
            len = length(scores_names)
            loop x from 0 to (len // 2):
                  # grouping the highest score with the lowest score
                  pairs.insert((scores_names[x], scores_names[len - 1]))
                  len = len – 1
            end loop
            pairs[0] = pairs[0] + [temp]
            create Tkinter button with 3 students in one group
            count = 1
            run func print_chart(count, len, col, row)

elif groupSize = 3:
      create list ‘triples’ to hold names of students in groups of 3
      if len % 3 = 0:
            loop x from 0 to (len // 3):
                  # grouping the highest score with the lowest score
                  triples.insert((scores_names[x], scores_names[len - 2],
            scores_names[len-1]))
                  len = len – 2
            end loop
            count = 0
            run func print_chart(count, len, col, row)

      elif len % 3 = 1:
            temp = scores_names[(len-1)//2]
            scores_names.pop[(len-1)//2]
            len = length(scores_names)
            loop x from 0 to (len // 3):
                  # grouping the highest score with the lowest score
                  triples.insert((scores_names[x], scores_names[len - 2],
            scores_names[len-1]))
                  len = len – 2
            end loop
            triples[0] = triples[0] + [temp]
            create Tkinter button with 4 students in one group
            count = 1
            run func print_chart(count, len, col, row)

      else:
            index = (len - 1) // 2
            temp, temp1 = scores_names [index], scores_names [index + 1]
```

```
            scores_names.pop(index)
            scores_names.pop(index + 1)
            len = length(scores_names)
            loop x from 0 to (len // 3):
                  # grouping the highest score with the lowest score
                  triples.insert((scores_names[x], scores_names[len - 2],
            scores_names[len-1]))
                  len = len – 2
            end loop
            # inserting group of 2
            triples.insert([temp, temp1])
            create Tkinter button with 2 students in one group
            count = 0
            run func print_chart(count, len, col, row)

elif groupSize = 4:
      create list ‘quadruple’ to hold names of students in groups of 4
      if len % 4 = 0:
            loop x from 0 to (len // 4):
                  # grouping the highest score with the lowest score
                  quadruple.insert((scores_names[x], scores_names[len - 3],
            scores_names[len-2], scores_names[len-1]))
                  len = len – 3
            end loop
            count = 0
            run func print_chart(count, len, col, row)

      elif len % 4 = 1:
            temp = scores_names[(len-1)//2]
            scores_names.pop[(len-1)//2]
            len = length(scores_names)
            loop x from 0 to (len // 4):
                  # grouping the highest score with the lowest score
                  quadruple.insert((scores_names[x], scores_names[len - 3],
            scores_names[len-2], scores_names[len-1]))scores_names[len-1]))
                  len = len – 3
            end loop
            quadruple[0] = quadruple[0] + [temp]
            create Tkinter button with 5 students in one group
            count = 1
            run func print_chart(count, len, col, row)

      elif len % 4 = 2:
            index = (len - 1) // 2
            temp, temp1 = scores_names [index], scores_names [index + 1]
            scores_names.pop(index)
            scores_names.pop(index + 1)
            len = length(scores_names)
            loop x from 0 to (len // 4):
                  # grouping the highest score with the lowest score
                  triples.insert((scores_names[x], scores_names[len - 2],
            scores_names[len-1]))
```

```
            len = len – 3
      end loop
      quadruple[0] = quadruple[0] + [temp]
      quadruple[1] = quadruple[1] + [temp1]
      create two Tkinter buttons to display the groups of 5 students
      count = 2
      run func print_chart(count, len, col, row)

elif len % 4 = 3:
      index = (len - 1) // 2
      temp, temp1, temp2 = scores_names[0], scores_names[index],
scores_names[len - 1]
      scores_names.pop(0)
      scores_names.pop(index)
      scores_names.pop(-1)
      len = length(scores_names)
      loop x from 0 to (len // 4):
            # grouping the highest score with the lowest score
            quadruple.insert((scores_names[x], scores_names[len - 3],
      scores_names[len-2], scores_names[len - 1]))
            len = len – 3
      end loop
      quadruple.insert([temp, temp1, temp2])
      create one Tkinter button to display the group of 3 students
      count = 0
      run func print_chart(count, len, col, row)
```

Test Table:

Action to Test	Success Criterion Tested	Method of Testing
Log-in functionality is working correctly for various users	#1, #1a, #1b, #12	• Attempt to log in with incorrect username and password to prompt error message.
		• Log in with correct username and password to prompt successful login message and be redirected to welcome screen.
Users can view and edit all student data which is simultaneously updated in the database	#2, #3, #4, #12	• Input data for a new student to prompt successful input message and scroll to the bottom of the table to see the insertion.
		• Update one student's data to prompt successful updation message and check if the same has updated in the table.
		• Delete one student's data to prompt successful deletion message and check if the same has been removed from the table.
Weighted scores are calculated correctly if student data is updated	#6	• Manually calculate one student's weighted score and match it against the displayed weighted score.
		• Update the same student's data and press 'Refresh Weighted Scores'. Recalculate the student's weighted score manually and match it against the new displayed weighted score.
Users are prompted to choose group size (only 2, 3, or 4) before generating seating chart	#7	• When 'Generate Seating Chart' is pressed, new window should open with drop down list of numeric options
Seating charts that are generated include all student names and are accurately divided according to weighted scores	#8, #10	• Match the names in the seating chart against names in the table
		• Check if the highest scorer is grouped with the lowest scorer
		• Print out a sorted list of grouped students and their weighted scores in the Python terminal and match it against the displayed seating chart
Seating charts take into odd/even number of students according to group size	#9	• Have an odd number of students in the class and attempt to generate a seating chart with students either divided into groups of 2 and 4. Seating chart should display the excess students as part of other groups in the class.
		• Have an even number of students in the class and attempt to generate a seating chart with students divided into groups of 3. Seating chart should display the excess students as part of other groups in the class.

Search feature is working correctly	#5	• Search for a value that does not exist and should display blank screen.
		• Search for an existing value and value entered should match the search displays.
Save feature is working correctly	#11	• Image saved should match the seating chart displayed in the program.
Button that clears all entry boxes	#13	• Enter values into the entry boxes and press clear entry boxes - all boxes should now be blank • Try by only putting a few values in different boxes (not all) and press the button