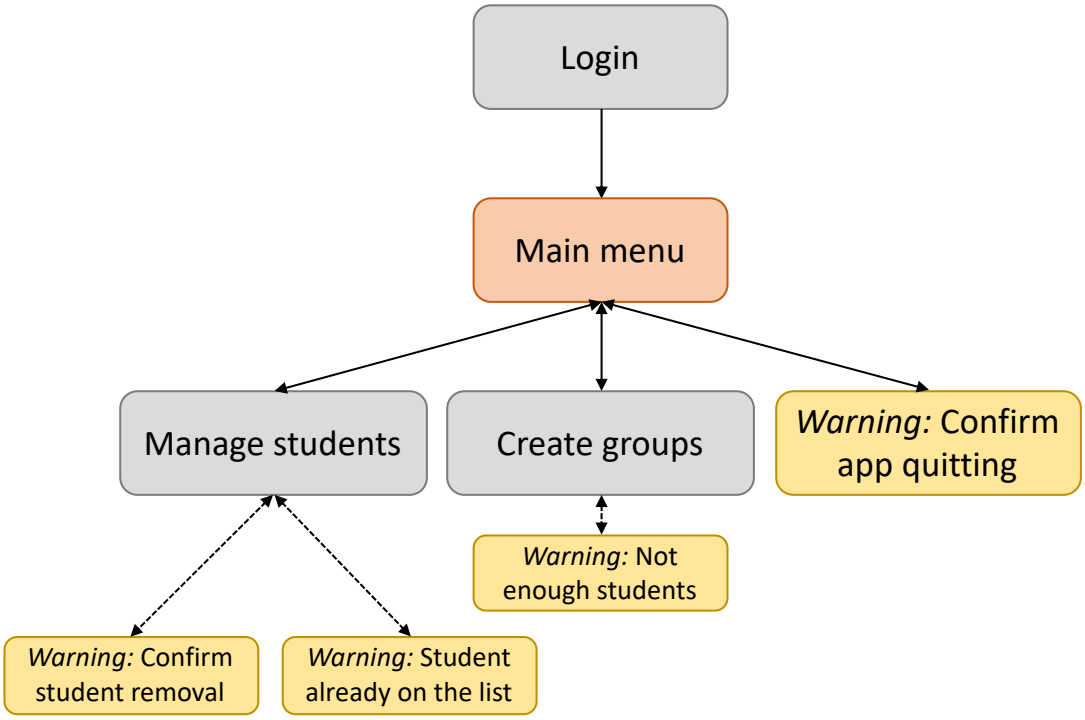
Criterion B: Design

Prototype solution

Screen hierarchy



Individual screens

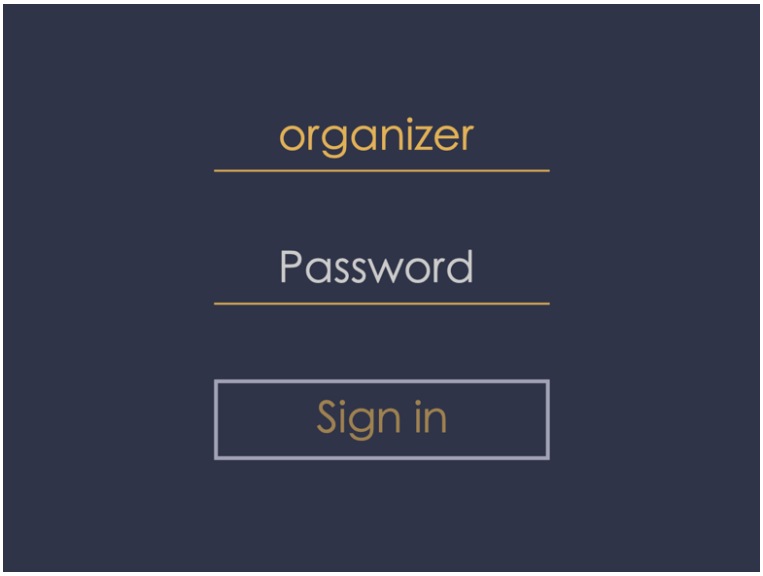


Figure 1: Login

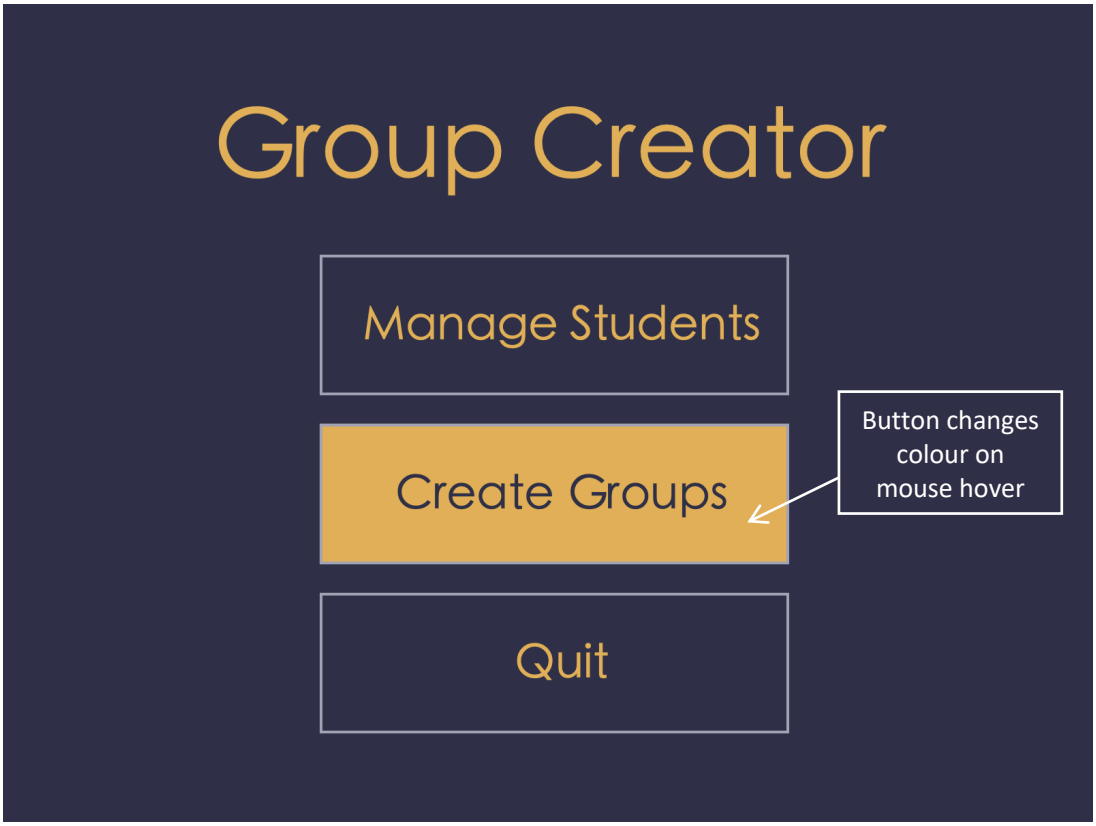


Figure 2: Main menu

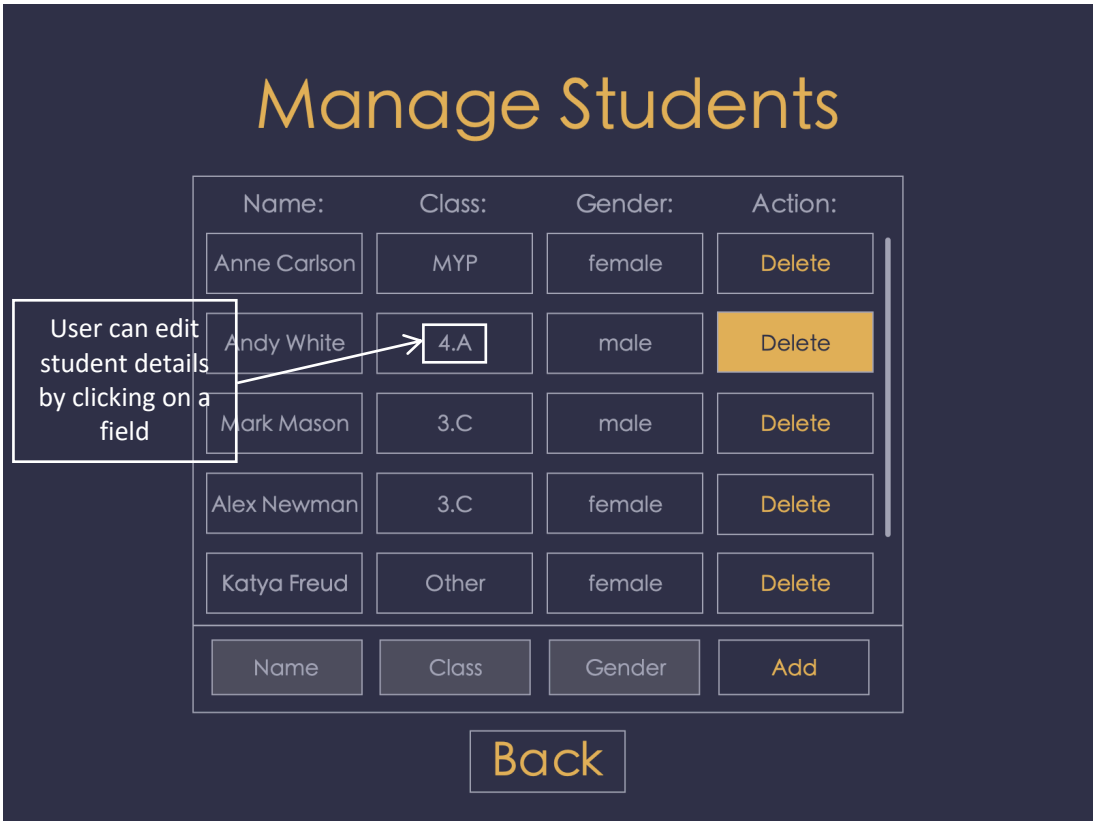


Figure 3: Manage students screen (for managing students, displaying relevant student details)

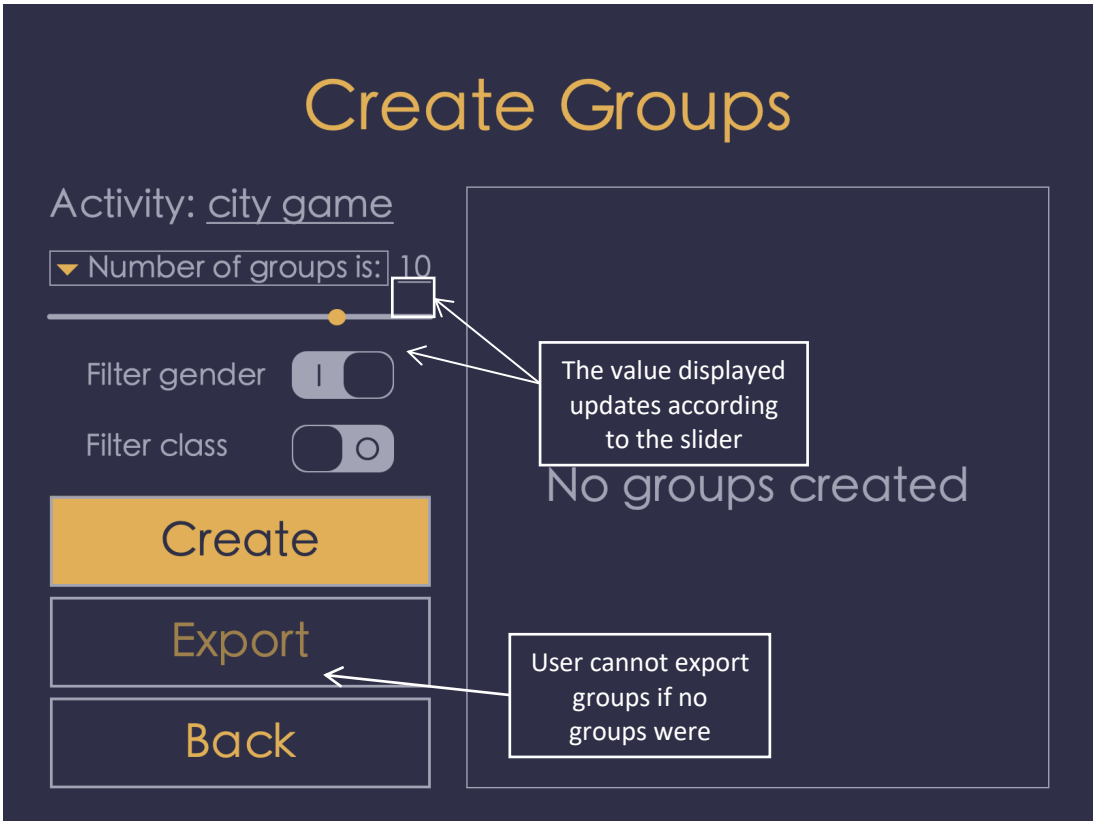


Figure 4: Create groups screen (before groups were created)

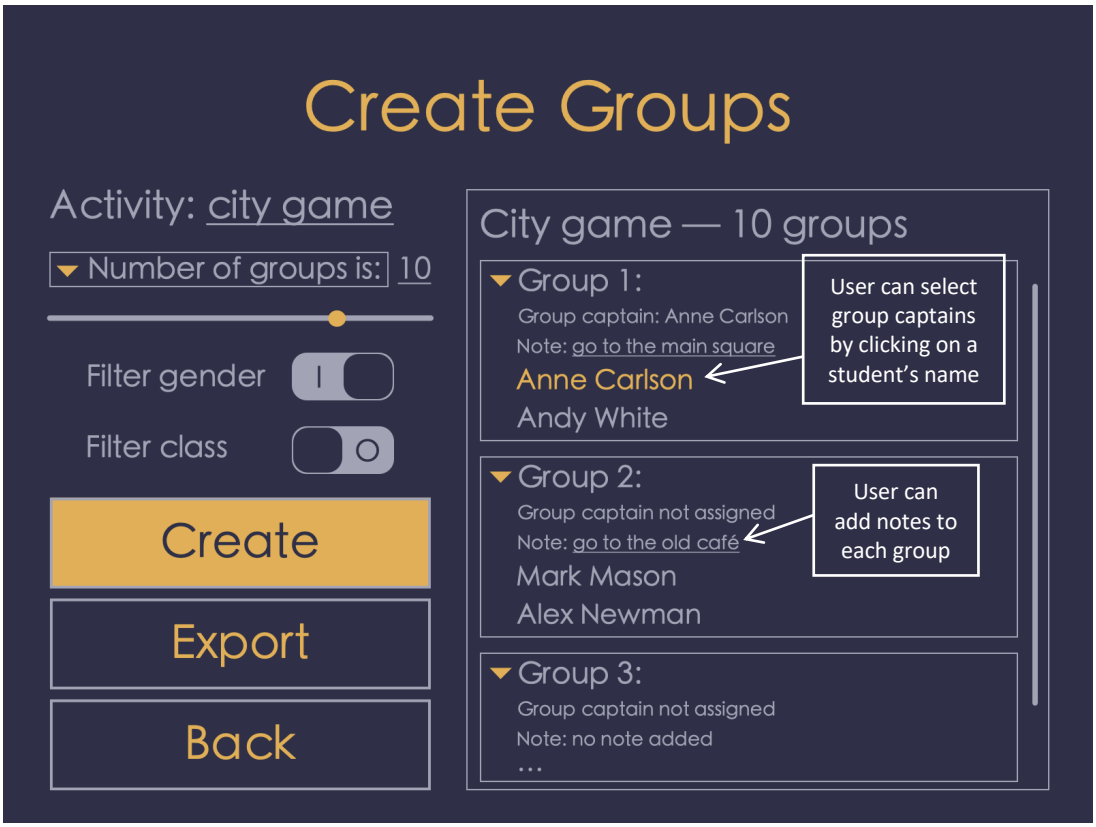


Figure 5: Create groups screen (after groups were created)

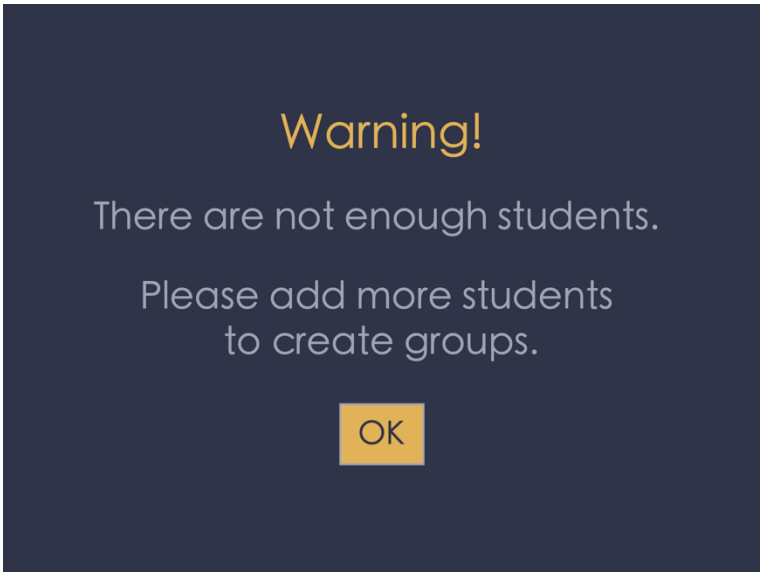


Figure 6: Warning: Not enough students

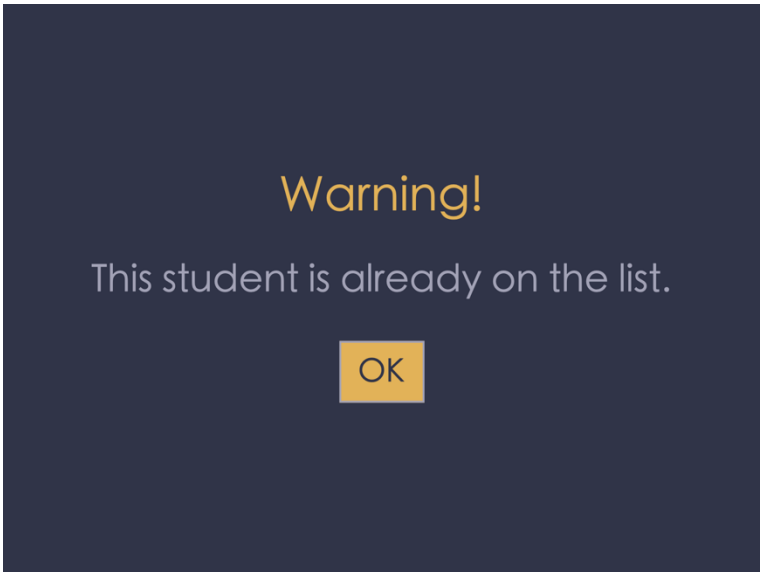


Figure 7: Warning: Student already on the list

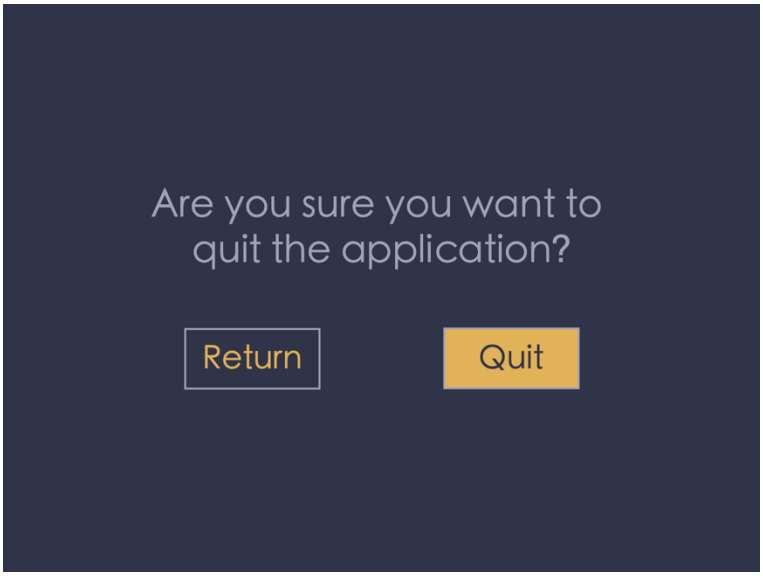


Figure 8: Warning: Confirm app quitting

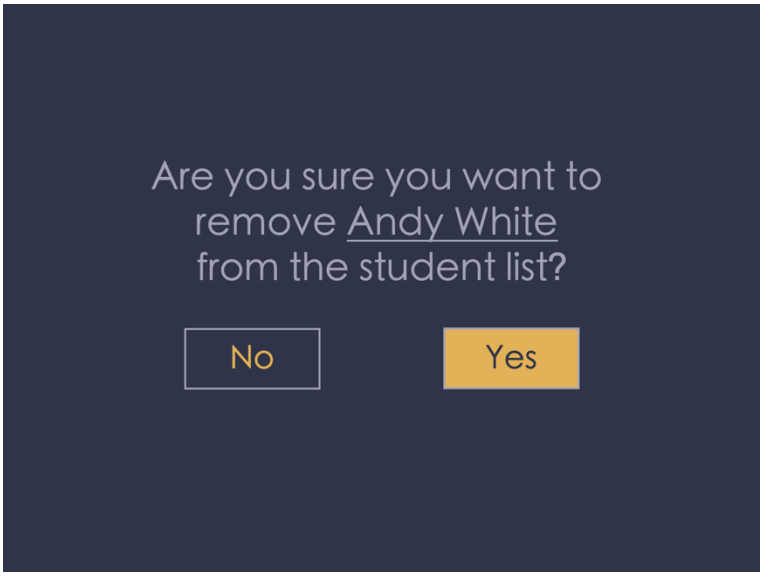


Figure 9: Warning: Confirm student removal

Data structures

Files

1. *File to store student data*

A plain, tab-delimited text file serving as a database.

```
male	MYP	Adam Gibbon
```

The data will be encrypted using Caesar cypher. The text file will look something like this:

```
ymxq	Ye\	Mpmy,Sunn{z
```

2. *File to store created groups*

A PDF file. The approximate format is the following:

```
Activity: Forest run
Group 1
        Leader: Fernando Earlywine
        Note: Go to the forest

        Students:
        Fernando Earlywine
        Roderick Kehrer
        Kirby Meza
```

Classes

Class name	Purpose
DataManager	Load and store data, handle operations on the student list.
Group	Represent one of the many generated groups.
GroupGenerator	Contain algorithms for group generation, export created groups into a PDF file.
MyDictionary	Hold class name as key and the number of students as value, provide information about when the list of classes is updated so that the change propagates to GUI.
Security	Hold username and password hash, provide user validation algorithms for login and decryption/encryption of strings when loading/storing data.
QuitApplication	Display alert and handle the event of application quitting.

StudentExists	Display alert.
CreateGroups	Display screen, handle events for numerous controllers specifying input criteria for group creation. Display a dynamic list of GUI elements representing groups after groups were created.
Login	Display screen, call validation algorithms in Security. Grant access to MainMenu.
MainMenu	Display screen.
ManageStudents	Display screen, handle operations on the student list (add/remove).
Main	Provide references to classes, switch between screens, close program properly.
Student	Represent a student, hold the fields name, previously attended class and gender.
StudentButton	Button that, if clicked, sets the leader of the associated group of students to the name currently displayed.

UML diagrams for key classes

Class Student	
- String name	Store the name of the student.
- String previousClass	Store the class previously attended by the student.
- String gender	Store the student's gender.
+ boolean isMale()	Return if the student is male. This will be used in some group creating algorithms.
… (access methods)	

Class Group	
- String[] members	Store students who belong in this group.
- String leader	Store the name of the group leader.
- String note	Store the note for this group.
- int groupNum	Store the group number.
- boolean leaderSet	Tells if the leader was set.
+ add(String name)	Adds a student to the members of the group.
… (access methods)	

Class DataManager	
- Student[] students	Store student objects.
- MyDictionary studentCounts	Store number of students in a class.
+ DataManager()	Initialise student list and list with student counts.
+ Student[] getStudents()	Return the array of students.
+ addStudent(String name, String previousClass, String gender)	Check if student is on student list. If not, add them.
+ removeStudent(Student s)	Remove a student from the student list.

+ loadData()	Load student data from a text file. Populate the student list.
+ storeData()	Store student data to the original text file.
+ getClasses()	Return a list of classes which the students previously attended.
+ getNumOfStudents()	Return number of students.

Class GroupGenerator	
- int numOfGroups	Store the number of groups to be created.
- Group[] groups	Store a list of group objects.
- String[] studentList	Store a list of student names in a specific order.
+ Group[] generate(int num, boolean isGroupsSize, boolean filterGender, boolean filterClass)	Calculates number of groups from group size if necessary, calls the correct group creation algorithm and returns a list of groups.
- algorithmA()	Each modifies the order of studentList based on a different combination of input criteria.
- algorithmB()	
- algorithmC()	
- algorithmD()	
- groupify()	Populate the list of groups groups using student names from the studentList, which is in a special order.
- export()	Create a text file and populate it with data from the list of groups.

Class MyDictionary	
+ boolean classAdded(String className)	Increment the number of students under className. If the class did not exist previously, return true.
+ boolean classRemoved(String className)	Decrement the number of students under className. If the class is now empty, return true.
+ boolean updateClass(String oldClass, newClass)	Decrement the number of students under oldClass, increment under newClass. If a class is created/removed, return true.
+ String[] getClassList()	Return a list of all classes.

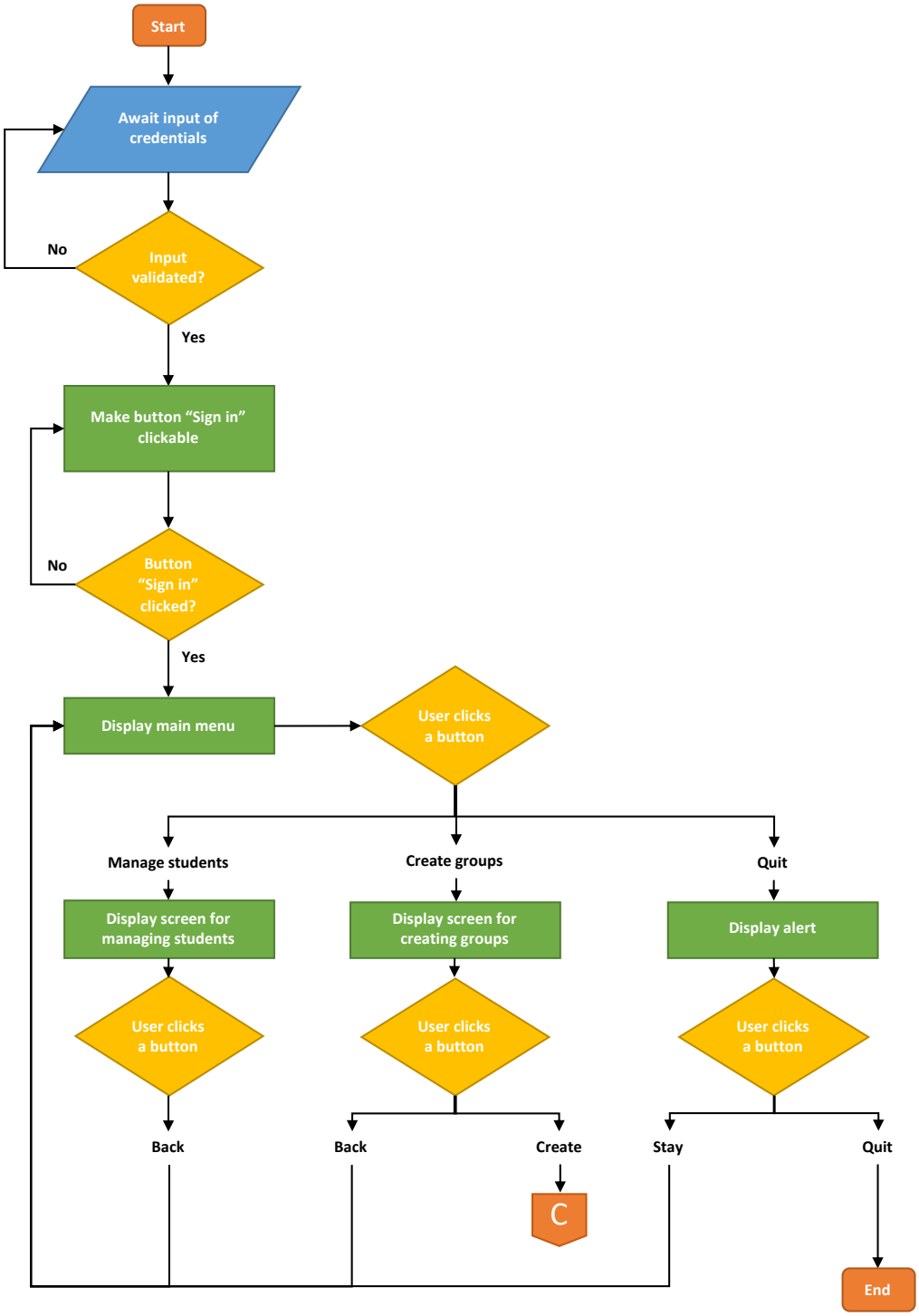
Class Security	
- String USER_NAME	Holds the only available user name.
- int PASSWORD	Holds a hash of the password.
- int KEY	Holds the key input by the user.
+ boolean verifyUsername(String username)	Return true if input username matches the one stored. Otherwise false.

+ boolean verifyPassword(String password)	Return true if the hash of input password matches the stored hash. Otherwise false.
+ boolean validateKey(String key)	Check if input key is in the proper format.
+ String encrypt(String s)	Encrypt and return the input string using Caesar cypher and the KEY.
+ String decrypt(String s)	Decrypt and return the input string using Caesar cypher and the KEY.

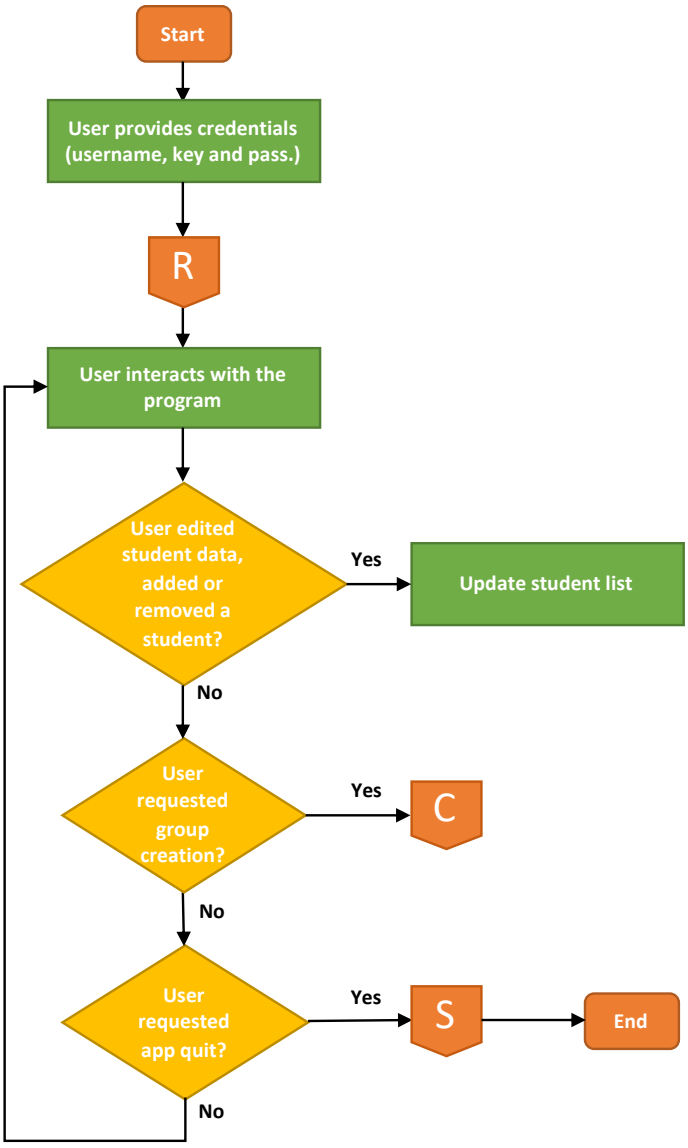
Algorithms

Flowcharts

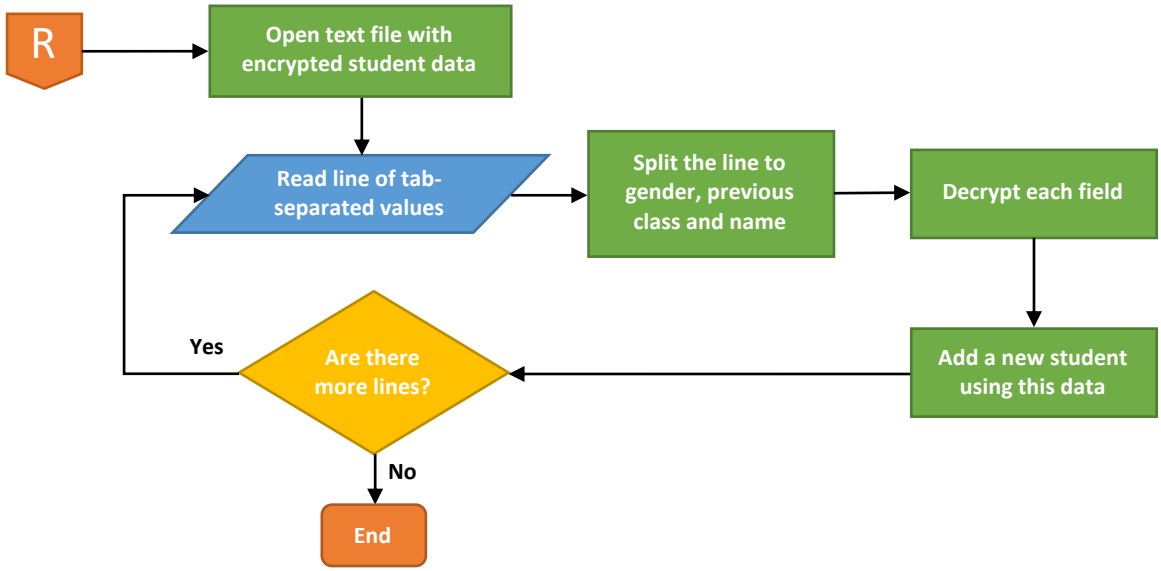
General program flowchart



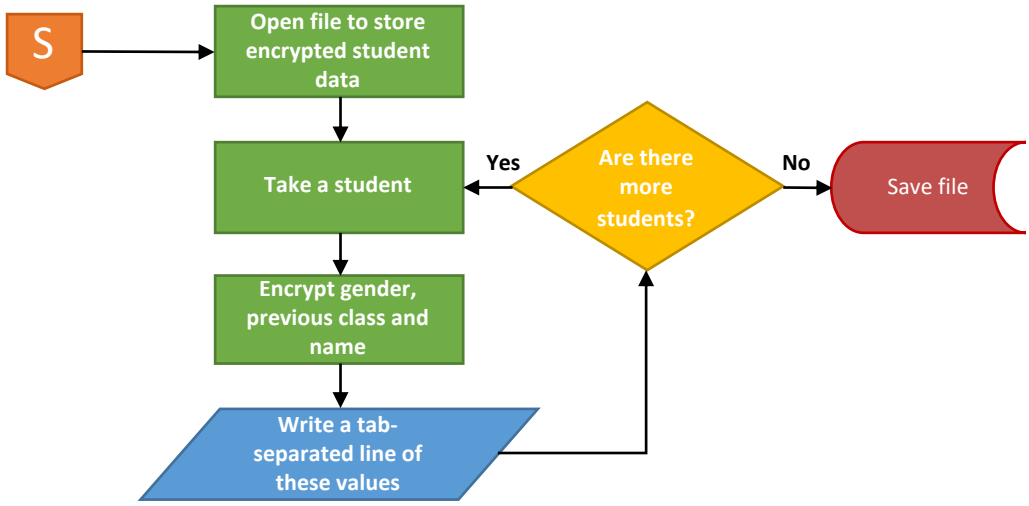
Data-focused program flowchart



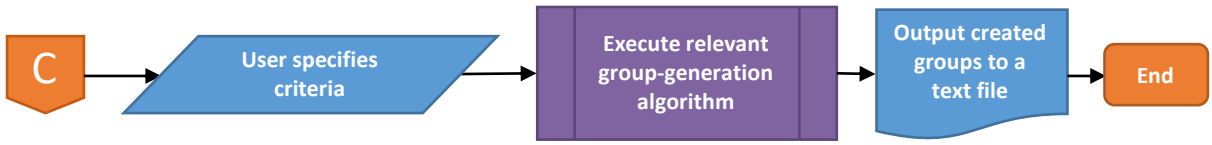
Reading student data from text file into a list of students in the program



Storing data to the text file containing student data



Creating groups in the program and outputting them to a file



Key algorithms

Group-generation algorithms:

Each of the below algorithms processes the student list into a specifically-ordered list of students to best match the user-defined criteria for creating groups.

1. Groups with same-class students dispersed

How it works: This algorithm creates a dictionary with class name as key and a list of student names as the value stored under that key. It randomises each list of student names for every class name and finally adds the student names from that list to a long list.

```
STUDENTS ← a list of all Student objects
STUDENTS_OF ← a dictionary containing for every class a list of student names
OUTPUT ← a list containing student names
for each STUDENT in STUDENTS do
    STUDENTS_OF[STUDENT.class].add(STUDENT.name)
end loop
for each CLASS in STUDENTS_OF.keys do
    shuffle(STUDENTS_OF[CLASS])
    for each NAME in STUDENTS_OF[CLASS]:
          OUTPUT.add(NAME)
    end loop
end loop
```

2. Groups with same-gender and same-class students dispersed

How it works: This algorithm creates a dictionary with class name as key and two lists – one for each gender available in the application – of student names as the value stored under that key. It randomises either list of student names for every class name and finally adds the student names from either list to a long list.

```
STUDENTS ← a list of all Student objects
STUDENTS_OF ← a dictionary containing for every class a list of student names
for either gender
OUTPUT ← a list containing student names
for each STUDENT in STUDENTS do
    if STUDENT.isMale = true then
          STUDENTS_OF[STUDENT.class][MALE].add(STUDENT.name)
    else
          STUDENTS_OF[STUDENT.class][FEMALE].add(STUDENT.name)
    end if
end loop
for each CLASS in STUDENTS_OF.keys do
    MALES ← a list containing all male student names
    FEMALES ← a list containing all female student names
    for each NAME in STUDENTS_OF[CLASS][MALE]:
          MALES.add(NAME)
    end loop
    shuffle(MALES)
    OUTPUT.add(MALES)
    for each NAME in STUDENTS_OF[CLASS][FEMALE]:
          FEMALES.add(NAME)
    end loop
    shuffle(FEMALES)
    OUTPUT.add(FEMALES)
end loop
```

3. Processing the list of student names

How it works: This algorithm takes the ordered array of student names and loops through all groups in a cyclical manner, adding each time one student to the current group, until all students have been added.

```
STUDENTS ← a list containing student names in an order specified by the group-generation algorithm
STUDENT_NUM ← current student number
STUDENT_NUM = 0
NUM_OF_STUDENTS ← total number of students
GROUPS ← a list of Group objects
GROUP_NUM ← current group number
GROUP_NUM = 0
NUM_OF_GROUPS ← total number of groups
while STUDENT_IDX < NUM_OF_STUDENTS do
    STUDENT = STUDENTS[STUDENT_NUM]
    GROUPS[GROUP_NUM].add(STUDENT)
    STUDENT_NUM = STUDENT_NUM + 1
    GROUP_NUM = (GROUP_NUM + 1) mod NUM_OF_GROUPS
end loop
```

Encryption

How it works: This algorithm is an implementation of the Caesar cipher. It works by moving each character a given number of characters further. For example, the character 'A' would with an offset 3 be changed to 'D'. During decryption, the value OFFSET is set to negative. So, to decrypt 'D' from the previous example, offset -3 would give the correct decrypted value of 'A'.

```
INPUT ← a string of characters for encryption
OUTPUT ← an encrypted string of characters
OFFSET ← an integer that tells by how many characters to shift the character
MAX_CHAR ← the maximum ASCII value of a character
for each CHARACTER in INPUT do
    OUTPUT = OUTPUT + (CHARACTER + OFFSET) mod MAX_CHAR
end loop
```

Word count: 236 words

Test plan for success criteria

Success criterion to be tested		Test method
UI is straightforward for the user		Discussion of proposed design with the client.
Student data data (name, gender, class) are protected from unauthorised access:		
	User must provide correct credentials to gain access to main menu	Try logging in with either a wrong username or password, or both. Ensure that inputting the correct ones gains access to the rest of the program.
	Information about students is unreadable when accessed from outside the application	Outside app: See the student database and determine if the data are readable.
		In app: Try entering an incorrect key for decryption and determine that the student data are decrypted incorrectly (*i.e.* unreadable), vice versa.
User can manage students (add and remove, or edit the associated fields):		
	User must fill in name, previously attended class and gender before adding a new student	Try adding a student without filling out the relevant fields. See if otherwise the student appears on the student list.
	User cannot add a student who is already on the list	Try adding a student with a name-class-gender combination that is already on the list. Ascertain that the student is not added.
	User can edit name, previously attended class and gender	Try editing all fields and see if the changes stay.
	User can remove students	Try removing a student. Observe if they are removed from the table.
	User's approval is necessary before removing a student	Await approval after requesting student removal.
Students' names contain diacritic and are displayed correctly		Try adding a student whose name contains special characters and editing a table cell to an entry containing special characters. Observe whether the added student's name or the edited cell's content changes respective to user input.
User can create groups by specifying group size or number of groups to be created, and whether students should be dispersed based on previously attended class, gender, or both:		
	User cannot create groups unless at least 2 students are added	Request "Create Groups" from main menu when no and only one student is on the student list. Await alert.
	User can create groups based on group size or number of groups	Try creating groups based on group size and number of groups.
	User cannot create groups larger than the actual number of students	See if the range of the slider allows to create groups larger than the number of students.
	User cannot create more groups than there are students	See if the range of the slider allows to create more groups than there are students.

	User can create groups based on gender and/or previously attended class, or neither	Try creating groups based on gender or previously attended class in various combinations and both and neither. See if the generated groups represent the current selection of criteria.
	User cannot export groups if no groups were created	Try exporting groups when no groups are created.
	User can view the groups created	View the panel for creating groups. Ascertain that all groups are displayed.
User can set the title of the activity for which groups are being created		Try setting the title of the activity.
Groups are created non-deterministically		Request group creation twice in a row with the same input criteria. See if the names of groups members change.
User can add notes and select a captain for each group after groups were created		Try adding a note and selecting and re-selecting group captains for multiple created groups.
User can export the activity name, created groups, notes and captains to a PDF file		Request exporting of the groups created. See the generated PDF file to confirm that activity title, group leaders, notes and the groups themselves are there.
Future users can access the product from an online repository		Try to access the project repository from a different computer.