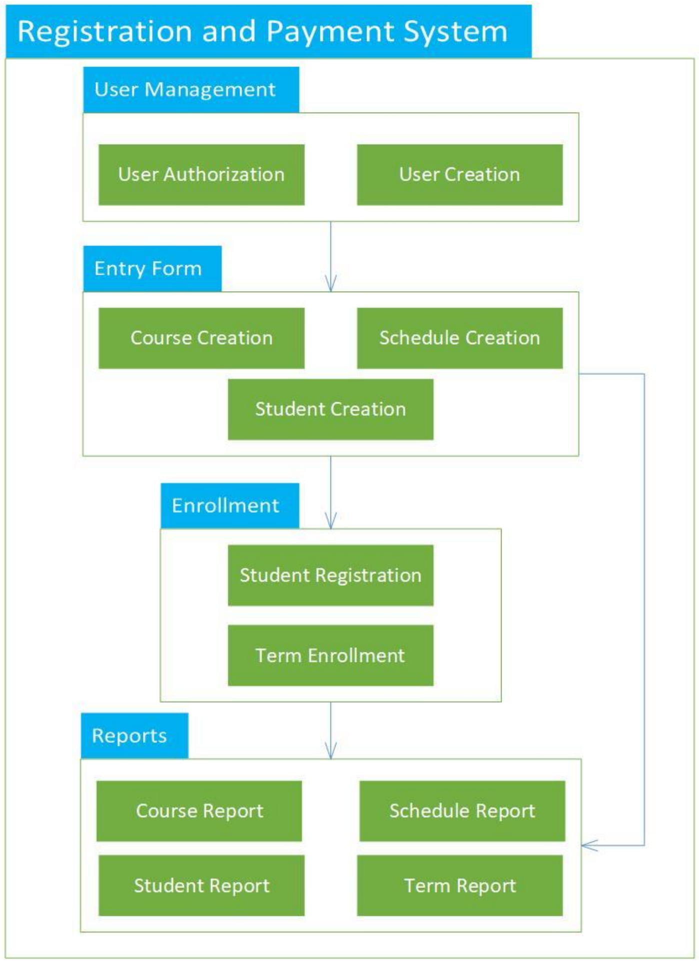
Table of Contents

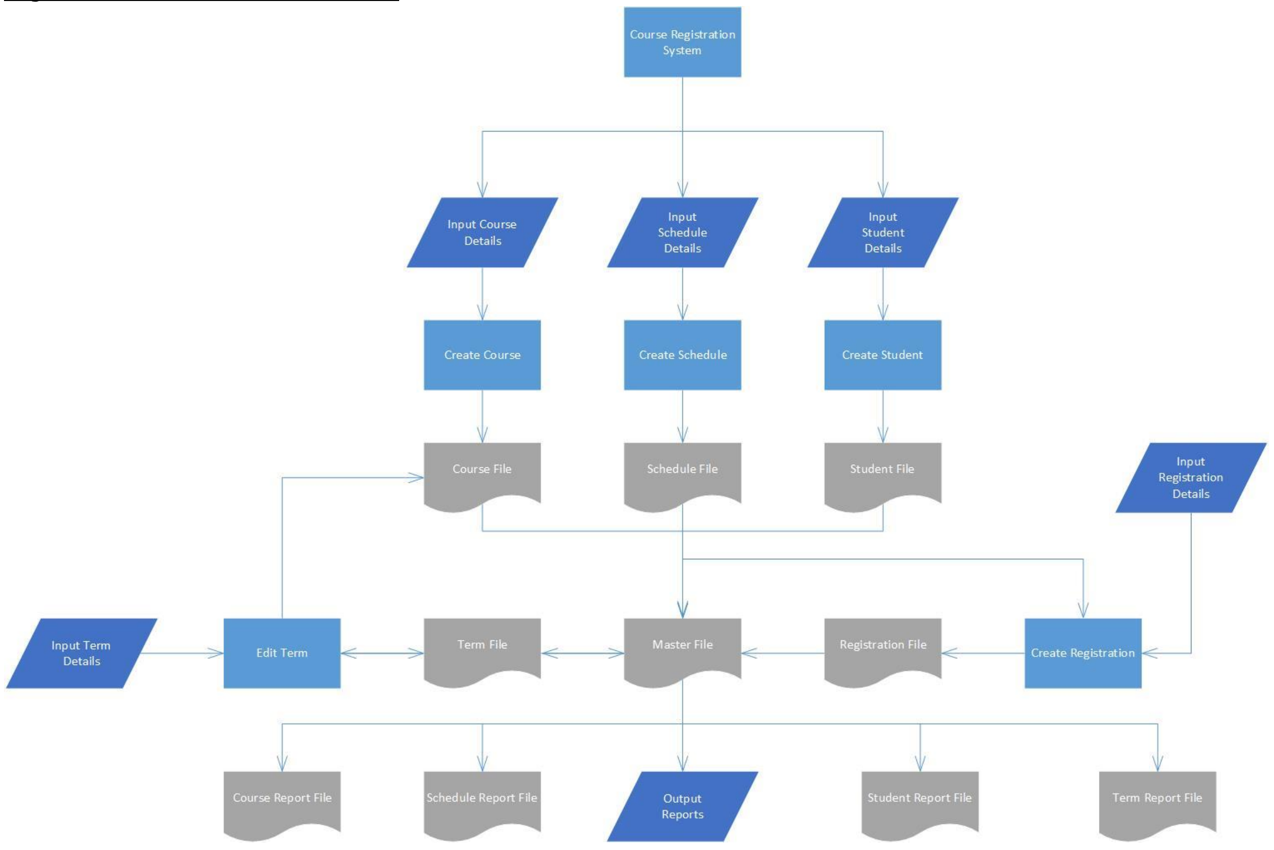
Structure Chart



Key

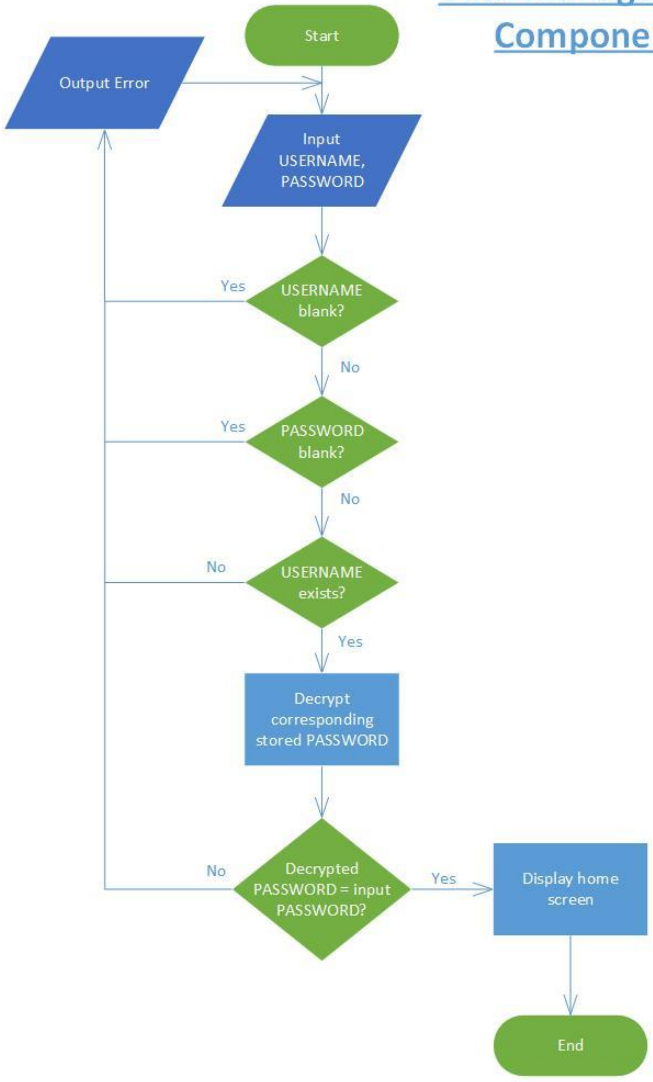
Component Title	🟦
Module	🟩

System Flow Chart

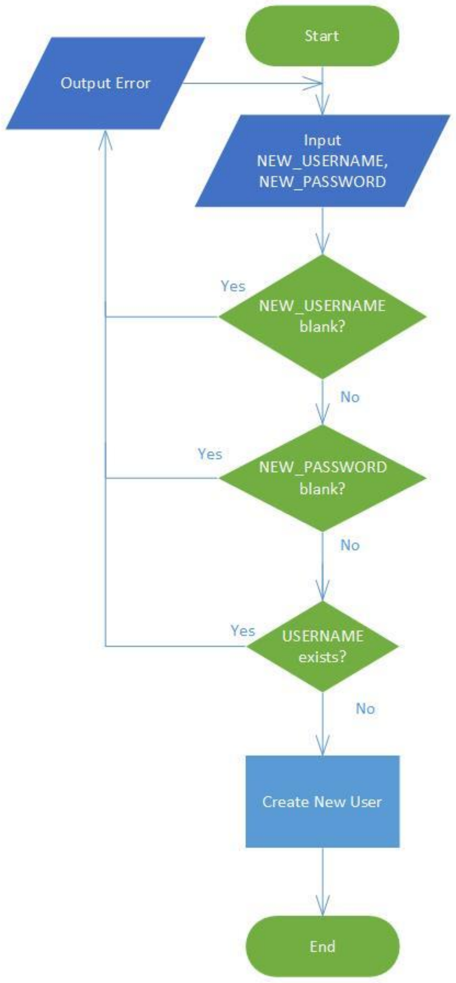


Product Flowchart

User Management Component

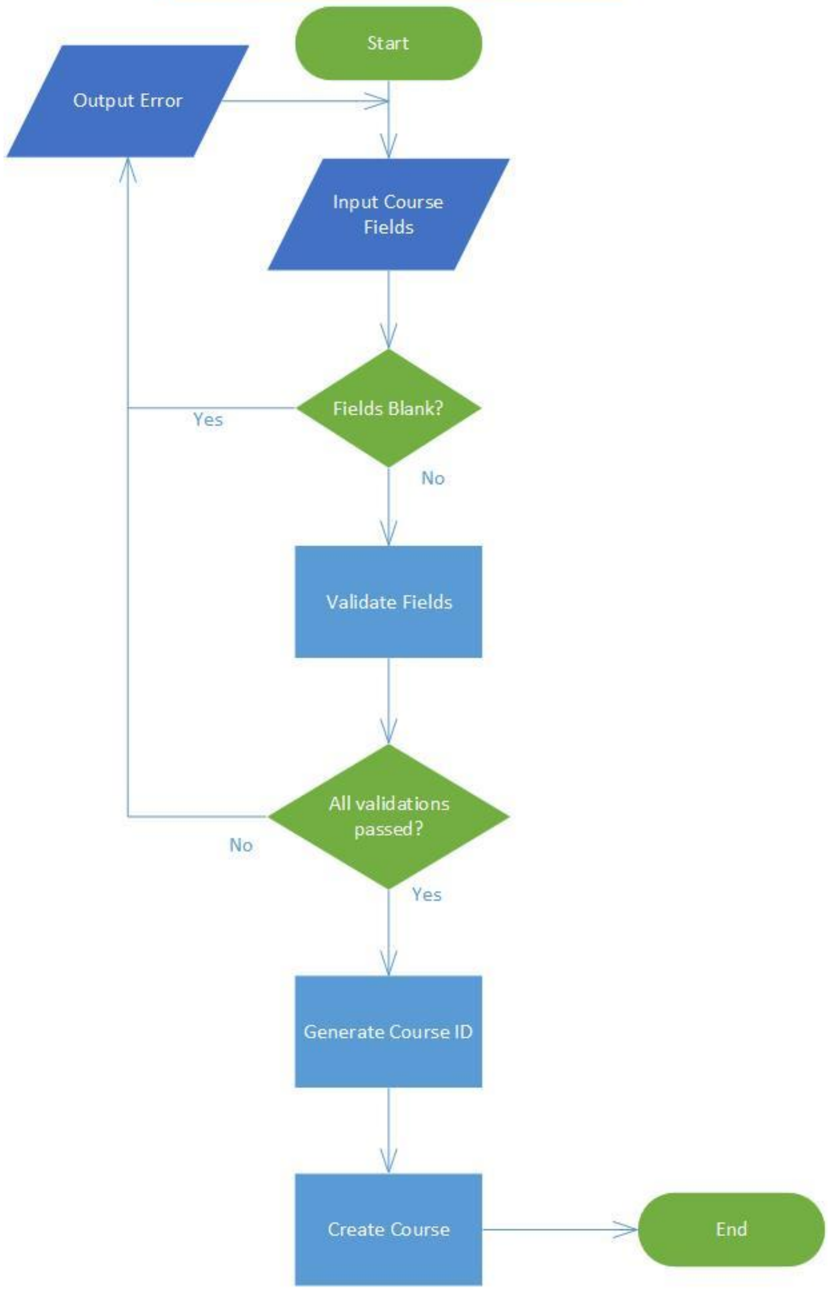


Flow for User Authorization



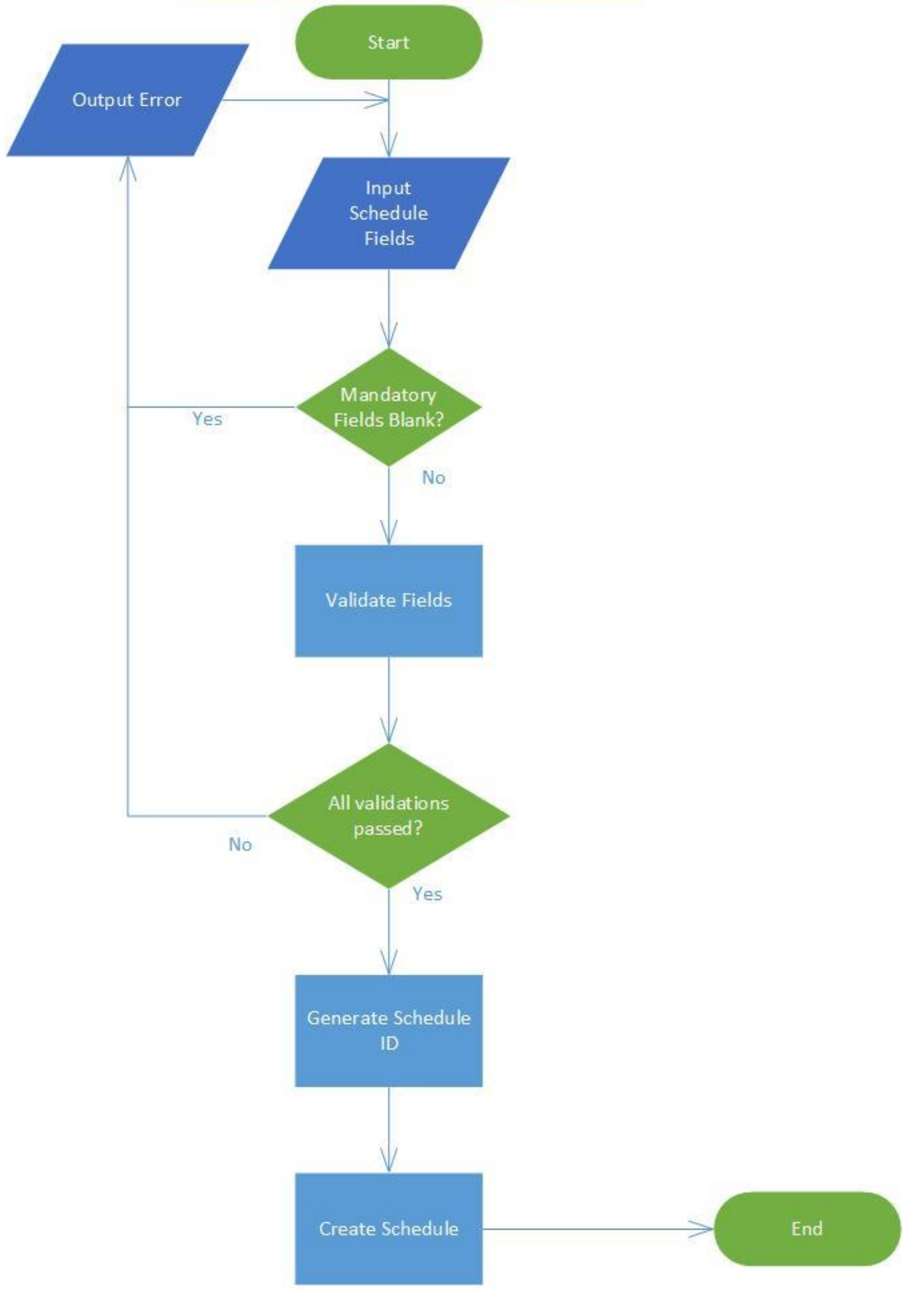
Flow for User Creation

Entry Form Component



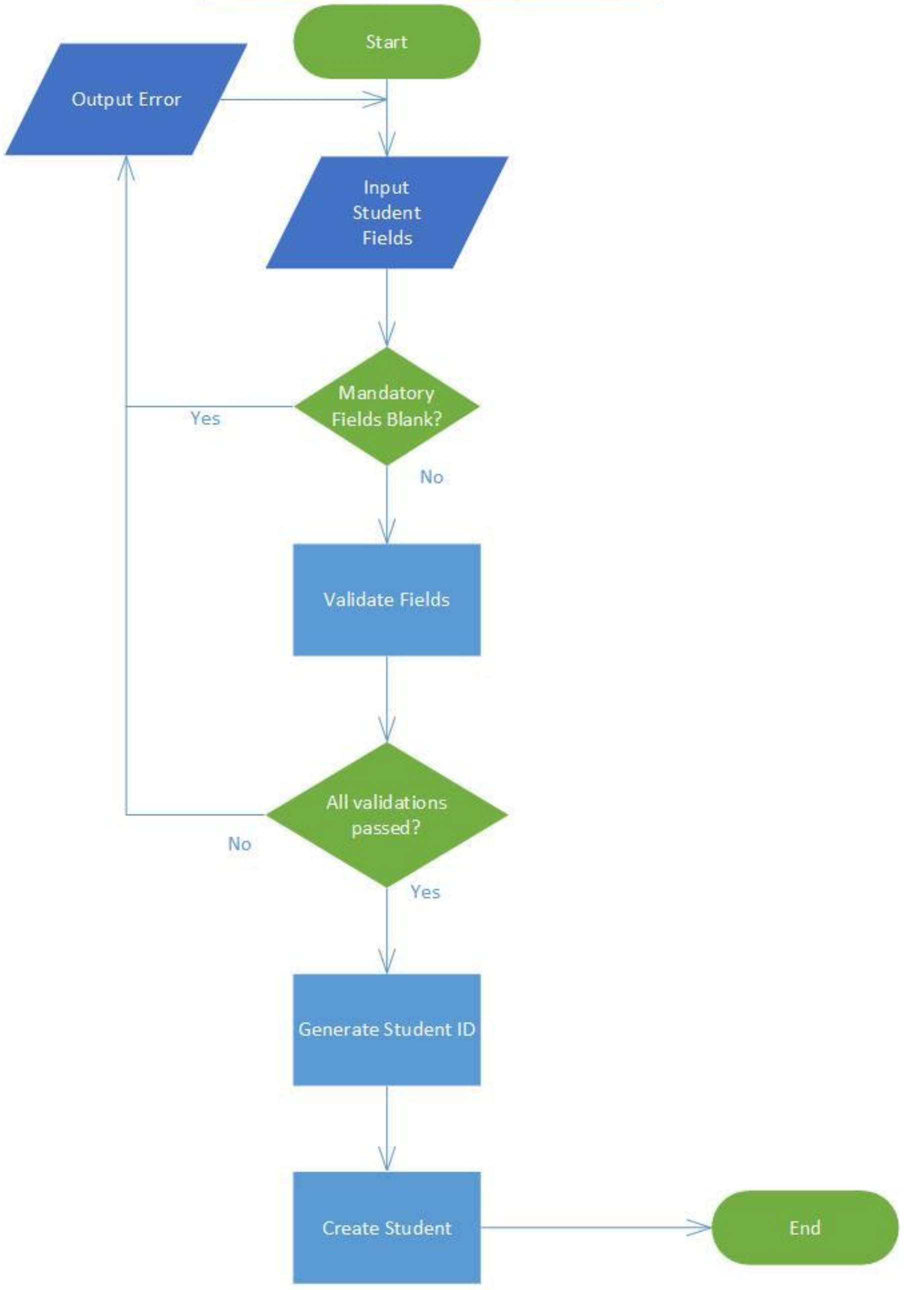
Flow for Course Creation

Entry Form Component



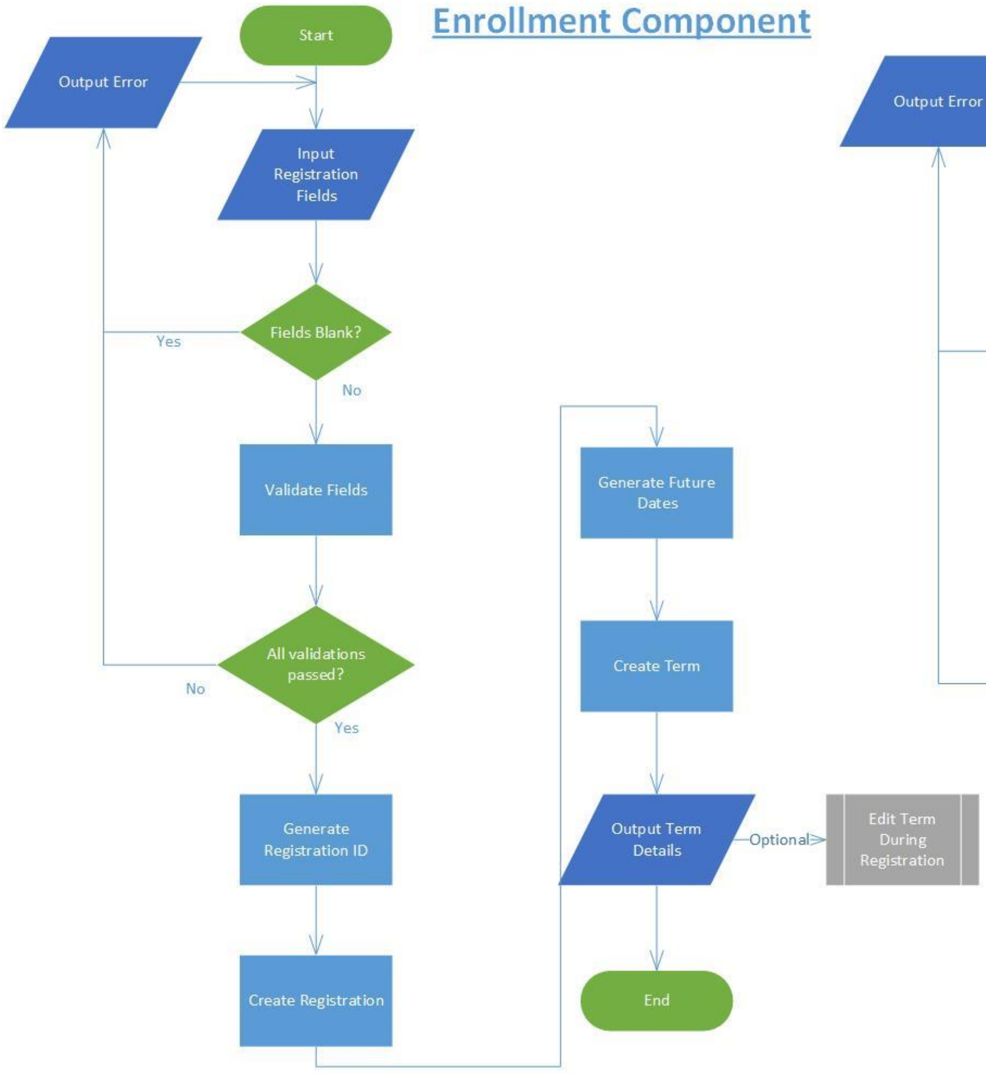
Flow for Schedule Creation

Entry Form Component

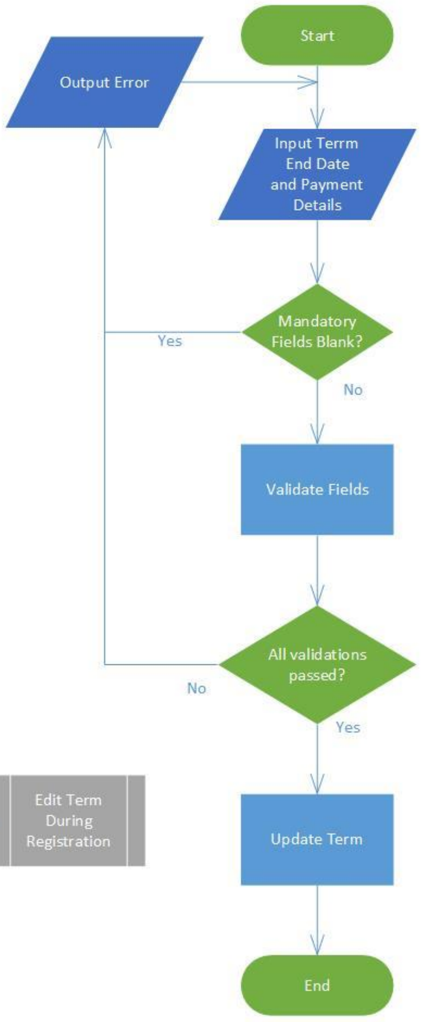


Flow for Student Creation

Enrollment Component

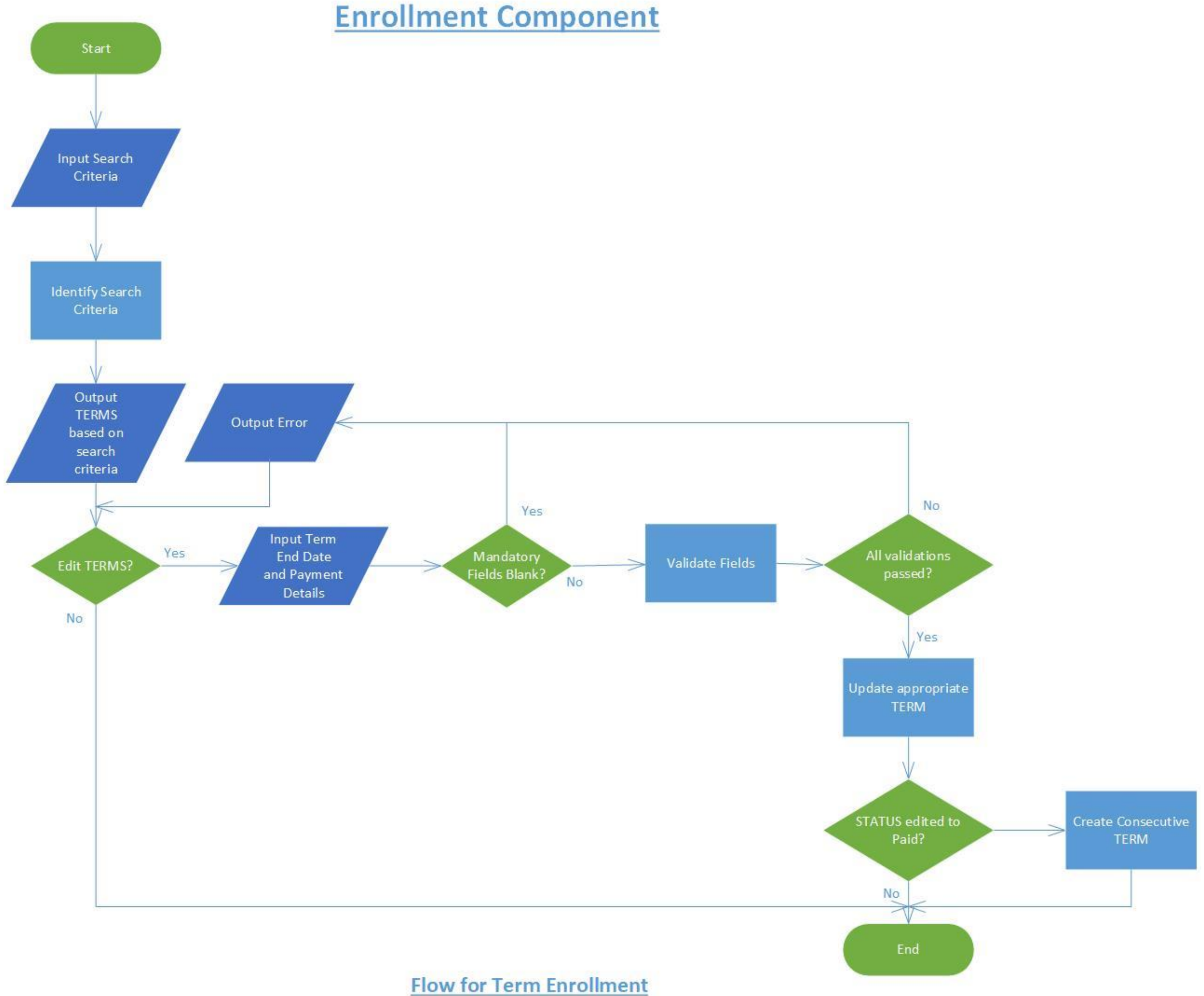


Flow for Student Registration



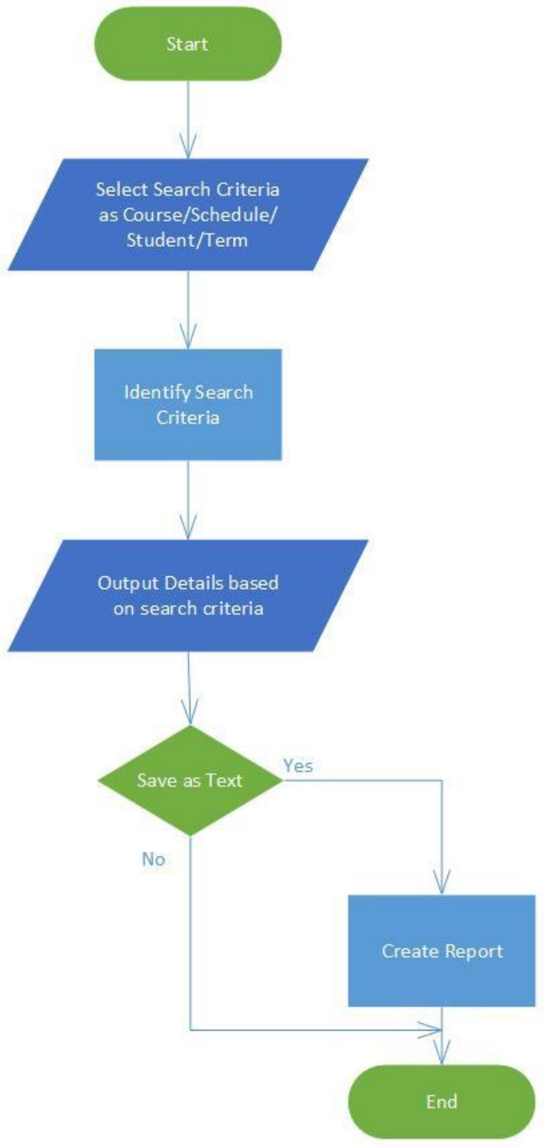
Flow for Edit Term Created During Registration

Enrollment Component



Flow for Term Enrollment

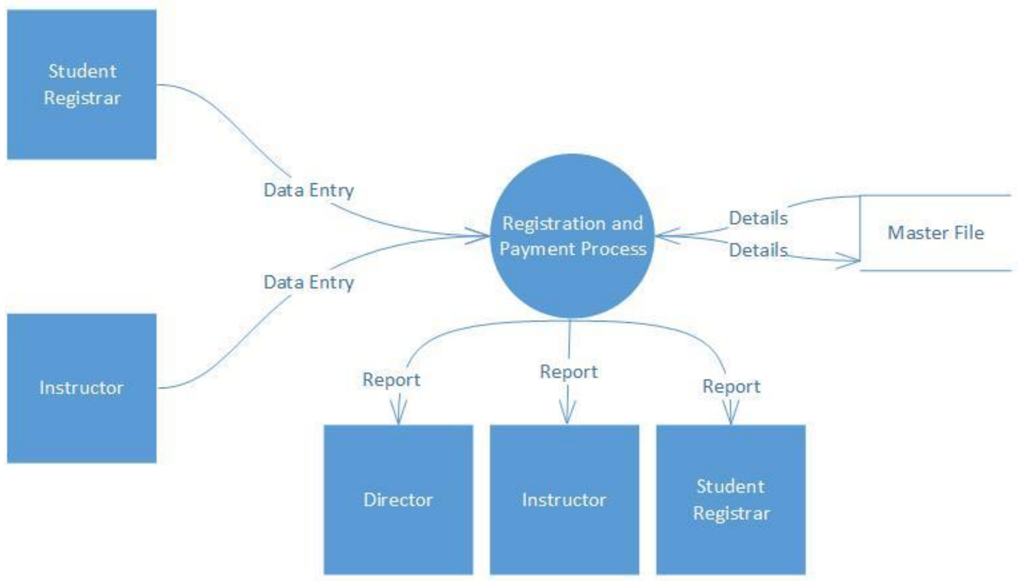
Report Component


Start
Select Search Criteria as Course/Schedule/ Student/Term
Identify Search Criteria
Output Details based on search criteria
Save as Text
Yes
No
Create Report
End

Flow for Report Generation of all Functions

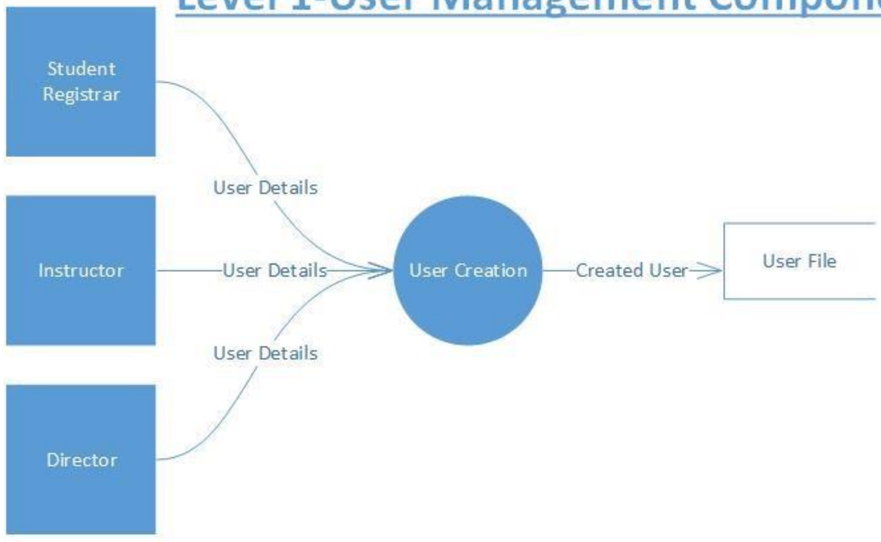
Data Flow Diagram

Level 0-Registration and Payment Process



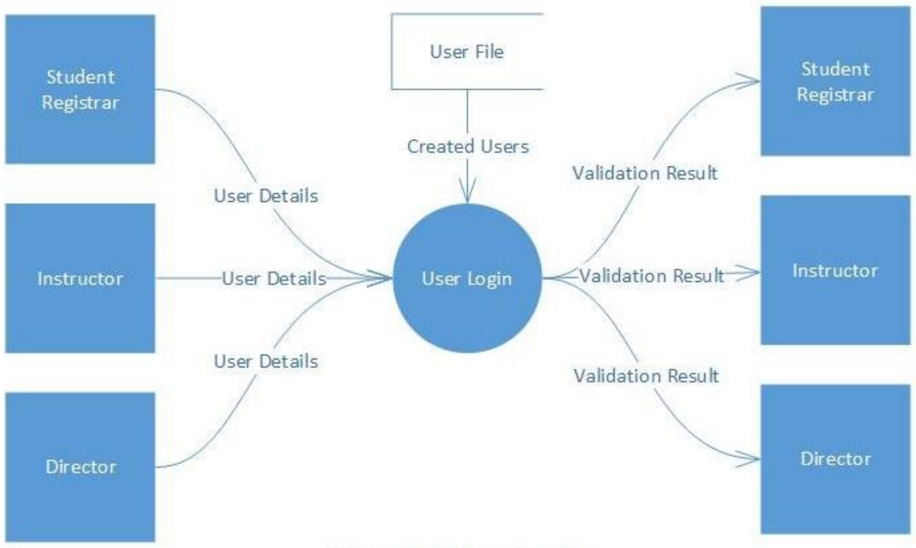
Overall Flow

Level 1-User Management Component



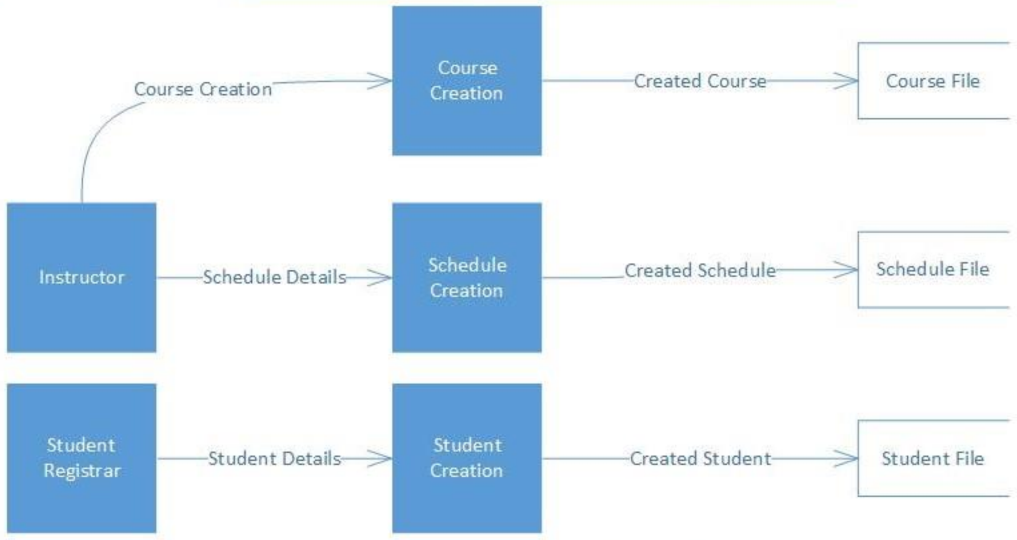
Flow for User Creation

Level 1-User Management Component



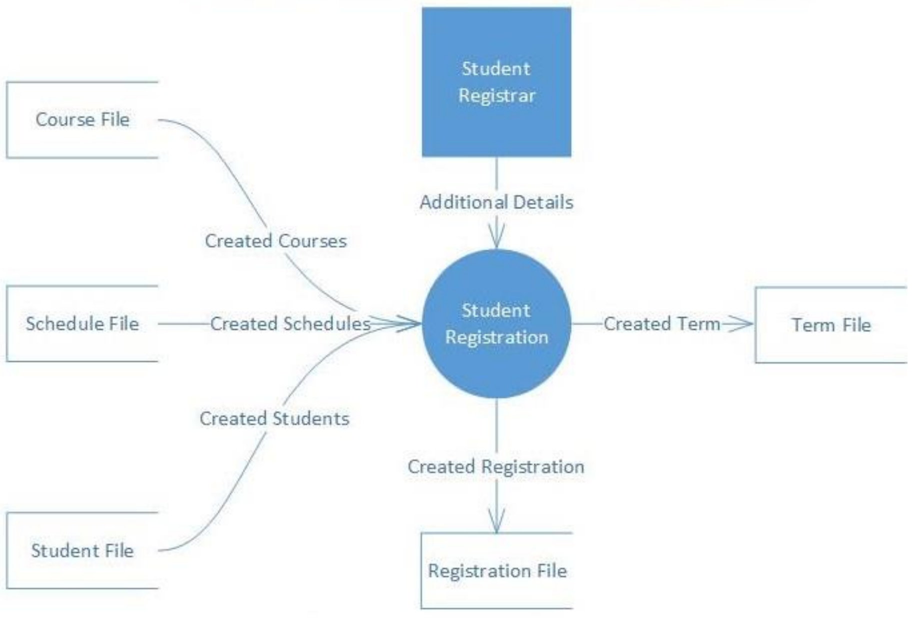
Flow for User Login

Level 1-Entry Form Component



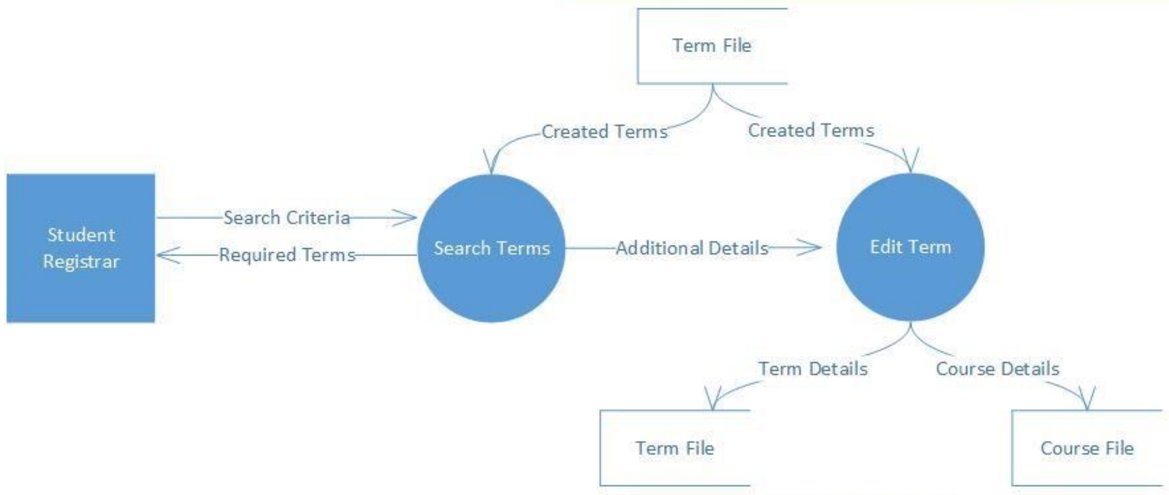
Flow for Entry Form Component

Level 1-Enrollment Component



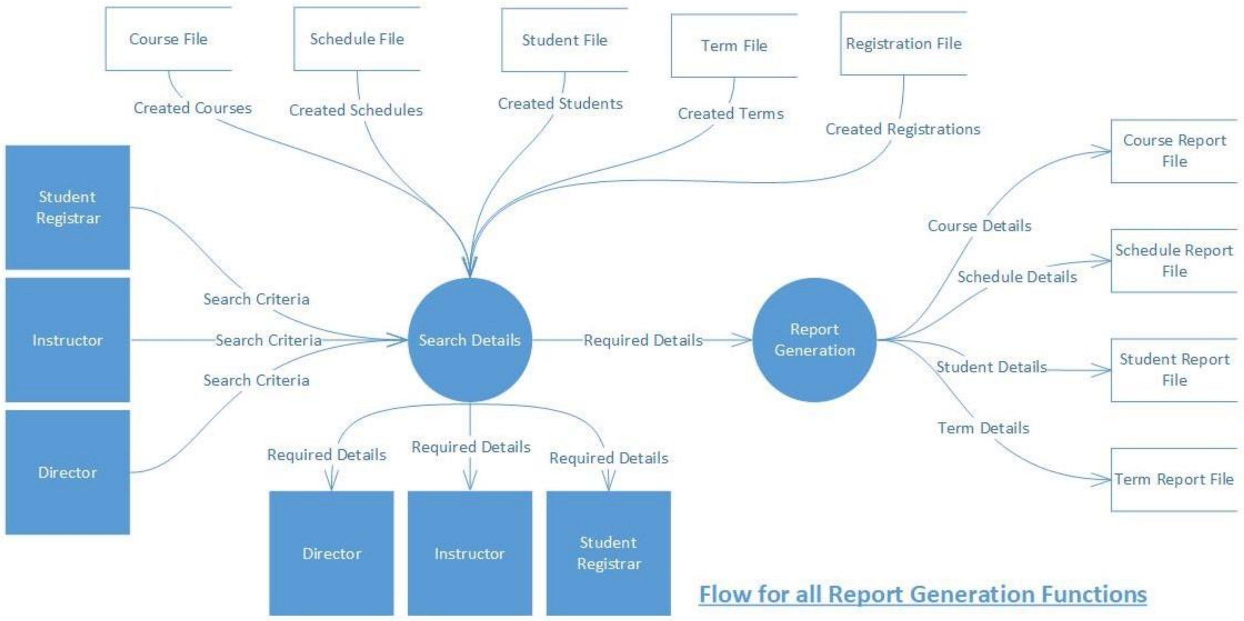
Flow for Student Registration

Level 1-Enrollment Component



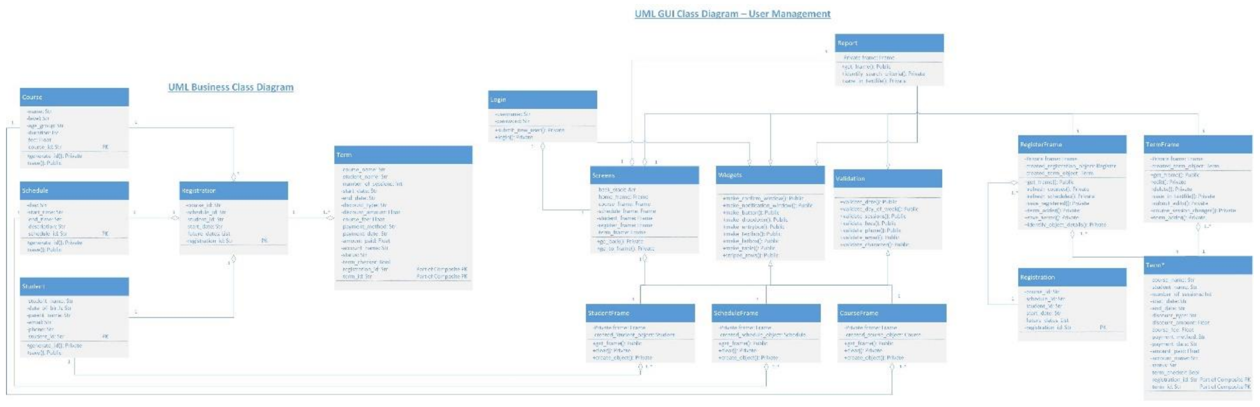
Flow for Term Enrollment

Level 1-Report Component



Flow for all Report Generation Functions

UML Class Diagram



Test Plan

Test ID	Test Type	Nature of Test	Expected Result
01	Create New Account	**Normal** **Test Case 1** Username*: user_name Password*: password	**Test Case 1** Successful User creation
		Abnormal **Test Case 2** Username*: (pre-existing username) Password*: password **Test Case 3** Username*: (blank) Password*: password **Test Case 4** Username*: username Password*: (blank)	**Test Case 2** Error message indicating username already exists **Test Case 3** (username not entered) Error message asking user to enter all required details **Test Case 4** (password not entered) Error message asking user to enter all required details
02	Log-in using existing Account	**Normal** **Test Case 1** Username*: user_name Password*: password	**Test Case 1** Successful Login. User granted access to system
		Abnormal **Test Case 2** Username*: name Password*: password **Test Case 3** Username*: user_name Password*: pass **Test Case 4** Username*: (blank) Password*: password	**Test Case 2** (username does not exist) Error message indicating username does not exist **Test Case 3** (password does not match) Error message indicating password do not match **Test Case 4**

		Test Case 5 Username*: username Password*: (blank)	(username not entered) Error message asking user to enter all required details **Test Case 5** (password not entered) Error message asking user to enter all required details
03	Enter Course Details	**Normal** **Test Case 1** Course Name*: Chess Online Regular Course Course Level*: Advanced Age Group*: 7-9 Sessions*: 8 Fees*: 100	**Test Case 1** Successful creation and persistence of created course
		Abnormal **Test Case 2** Course Name*: Chess Online Regular Course Course Level*: Advanced Age Group*: 7-9 Sessions*: hello Fees*: 100 **Test Case 3** Course Name*: Chess Online Regular Course Course Level*: Advanced Age Group*: 7-9 Sessions*: 7.8 Fees*: 100 **Test Case 4** Course Name*: Chess Online Regular Course Course Level*: Advanced Age Group*: 7-9 Sessions*: 8 Fees*: hello	**Test Case 2** (invalid session – string) Error message indicating invalid data entry and asking user to enter correct data **Test Case 3** (invalid session – decimal value) Error message indicating invalid data entry and asking user to enter correct data **Test Case 4** (invalid fees) Error message indicating invalid data entry and asking user to enter correct data

		Test Case 5 Course Name*: (blank) Course Level*: Advanced Age Group*: 7-9 Sessions*: 8 Fees*: 100 **Test Case 6** Course Name*: Chess Online Regular Course Course Level*: (blank) Age Group*: 7-9 Sessions*: 8 Fees*: 100 **Test Case 7** Course Name*: Chess Online Regular Course Course Level*: Advanced Age Group*: (blank) Sessions*: 8 Fees*: 100 **Test Case 8** Course Name*: Chess Online Regular Course Course Level*: Advanced Age Group*: 7-9 Sessions*: (blank) Fees*: 100 **Test Case 9** Course Name*: Chess Online Regular Course Course Level*: Advanced Age Group*: 7-9 Sessions*: 8 Fees*: (blank)	**Test Case 5** (course name not entered) Error message asking user to enter all required details **Test Case 6** (course level not entered) Error message asking user to enter all required details **Test Case 7** (age group not entered) Error message asking user to enter all required details **Test Case 8** (sessions not entered) Error message asking user to enter all required details **Test Case 9** (fees not entered) Error message asking user to enter all required details
04		**Normal**	

	Enter Schedule Details	**Test Case 1** Day*: Monday Start Time*: 0830 End Time*: 1000 Description: None	**Test Case 1** Successful creation and persistence of created schedule
		Abnormal **Test Case 2** Day*: Monday Start Time*: 1130 End Time*: 1000 Description: None **Test Case 3** Day*: (blank) Start Time*: 0830 End Time*: 1000 Description: None **Test Case 4** Day*: Monday Start Time*: (blank) End Time*: 1000 Description: None **Test Case 5** Day*: Monday Start Time*: 0830 End Time*: (blank) Description: None	**Test Case 2** (start time > end time) Error message indicating invalid data entry and asking user to enter correct data **Test Case 3** (days not entered) Error message asking user to enter all required details **Test Case 4** (start time not entered) Error message asking user to enter all required details **Test Case 5** (end time not entered) Error message asking user to enter all required details
05	Enter Student Details	**Normal** **Test Case 1** Student Name*: Alex Smith Date of Birth*: 10/01/2014 Parent Name*: Peter Smith	**Test Case 1** Successful creation and persistence of created student

		Email: petersmith@gmail.com Phone Number: 91234567	
		Abnormal **Test Case 2** Student Name*: Alex Smith Date of Birth*: 31/02/2014 Parent Name*: Peter Smith Email: petersmith@gmail.com Phone Number: 91234567 **Test Case 3** Student Name*: Alex Smith Date of Birth*: 10/01/2014 Parent Name*: Peter Smith Email: hello Phone Number: 91234567 **Test Case 4** Student Name*: Alex Smith Date of Birth*: 10/01/2014 Parent Name*: Peter Smith Email: petersmith@gmail.com Phone Number: hello **Test Case 5** Student Name*: (blank) Date of Birth*: 10/01/2014 Parent Name*: Peter Smith Email: petersmith@gmail.com Phone Number: 91234567 **Test Case 6** Student Name*: Alex Smith Date of Birth*: (blank) Parent Name*: Peter Smith	**Test Case 2** (invalid date – doesn't exist) Error message indicating invalid data entry and asking user to enter correct data **Test Case 3** (invalid email) Error message indicating invalid data entry and asking user to enter correct data **Test Case 4** (invalid phone) Error message indicating invalid data entry and asking user to enter correct data **Test Case 5** (student name not entered) Error message asking user to enter all required details **Test Case 6** (date of birth not entered) Error message asking user to enter all required details

		Email: petersmith@gmail.com Phone Number: 91234567 **Test Case 7** Student Name*: Alex Smith Date of Birth*: 10/01/2014 Parent Name*: (blank) Email: petersmith@gmail.com Phone Number: 91234567	**Test Case 7** (parent name not entered) Error message asking user to enter all required details
06	Enter Registration Details	**Normal** **Test Case 1** Course*: {Chess Online Regular course, Advanced, 7-9} Student*: Alex Smith Schedule*: {Monday, 0830-1000} Start Date*: 17/01/2022	**Test Case 1** Successful creation and persistence of created registration. Automatic term generation and persistence.
		Abnormal **Test Case 2** Course*: {Chess Online Regular course, Advanced, 7-9} Student*: Alex Smith Schedule*: {Monday, 0830-1000} Start Date*: 15/01/2022 **Test Case 3** Course*: (blank) Student*: Alex Smith Schedule*: {Monday, 0830-1000} Start Date*: 17/01/2022 **Test Case 4** Course*: {Chess Online Regular course, Advanced, 7-9} Student*: (blank)	**Test Case 2** (start date is not the same day of the week as presented in schedule) Error message indicating invalid data entry and asking user to enter correct data **Test Case 3** (course not entered) Error message asking user to enter all required details **Test Case 4** (student not entered) Error message asking user to enter all required details

		Schedule*: {Monday, 0830-1000} Start Date*: 17/01/2022 **Test Case 5** Course*: {Chess Online Regular course, Advanced, 7-9} Student*: Alex Smith Schedule*: (blank) Start Date*: 17/01/2022 **Test Case 6** Course*: {Chess Online Regular course, Advanced, 7-9} Student*: Alex Smith Schedule*: {Monday, 0830-1000} Start Date*: (blank)	**Test Case 5** (schedule not entered) Error message asking user to enter all required details **Test Case 6** (start date not entered) Error message asking user to enter all required details
07	Search Students	**Normal** **Test Case 1** Search Criterion: Alex	**Test Case 1** Students Named 'Alex' displayed during the registration process
		Abnormal **Test Case 2** Search Criterion: October	**Test Case 2** No student displayed as no such student exists.
08	Enter Term Details	**Normal** **Test Case 1** End Date*: 14/02/2022 Discount Type: Sibling Discount Amount: 14 Payment Method: Bank Transfer Amount Paid: 86 Payment Date: 10/01/2022 Account Name: SmithPeter	**Test Case 1** Successful edit and persistence of last created term

		Status*: Pending	
		Abnormal **Test Case 2** End Date*: 31/02/2022 Discount Type: Sibling Discount Amount: 14 Payment Method: Bank Transfer Amount Paid: 86 Payment Date: 10/01/2022 Account Name: SmithPeter Status*: Pending **Test Case 3** End Date*: 14/02/2022 Discount Type: Sibling Discount Amount: hello Payment Method: Bank Transfer Amount Paid: 86 Payment Date: 10/01/2022 Account Name: SmithPeter Status*: Pending **Test Case 4** End Date*: 14/02/2022 Discount Type: Sibling Discount Amount: 14 Payment Method: Bank Transfer Amount Paid: hello Payment Date: 10/01/2022 Account Name: SmithPeter Status*: Pending **Test Case 5** End Date*: 14/02/2022	**Test Case 2** (invalid end date) Error message indicating invalid data entry and asking user to enter correct data **Test Case 3** (invalid discount amount) Error message indicating invalid data entry and asking user to enter correct data **Test Case 4** (invalid amount paid) Error message indicating invalid data entry and asking user to enter correct data **Test Case 5**

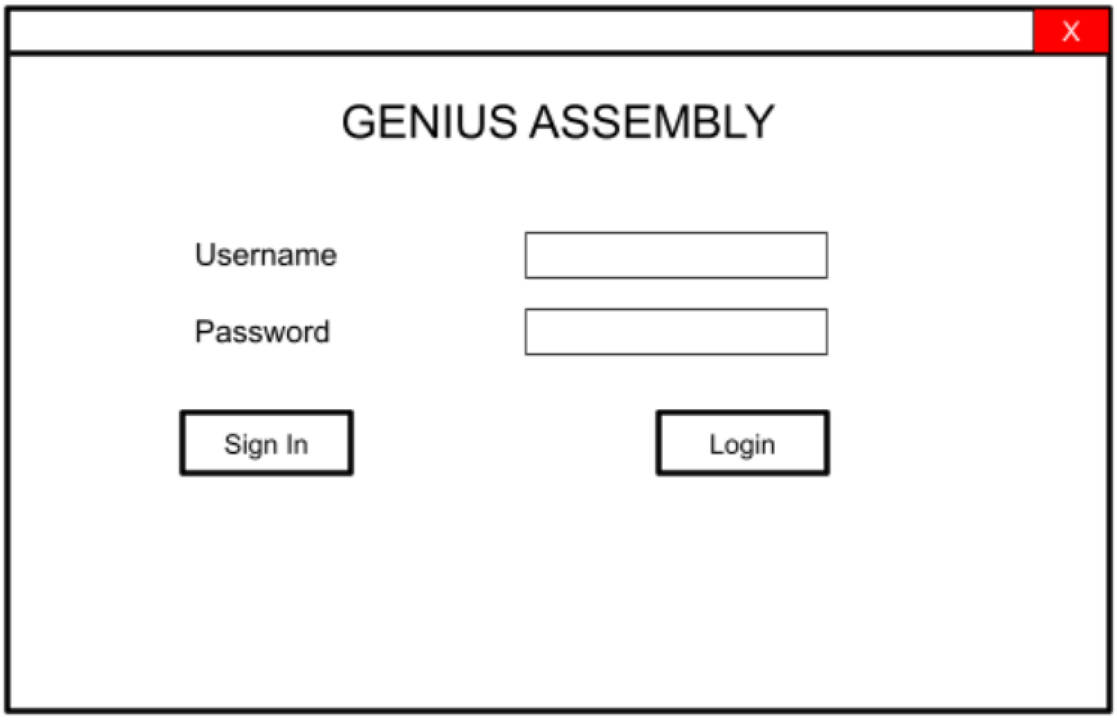
		Discount Type: Sibling Discount Amount: 14 Payment Method: Bank Transfer Amount Paid: 86 Payment Date: 41/01/2022 Account Name: SmithPeter Status*: Pending **Test Case 6** End Date*: (blank) Discount Type: Sibling Discount Amount: 14 Payment Method: Bank Transfer Amount Paid: 86 Payment Date: 10/01/2022 Account Name: SmithPeter Status*: Pending	(invalid payment date) Error message indicating invalid data entry and asking user to enter correct data **Test Case 6** (end date not entered) Error message indicating invalid data entry and asking user to enter correct data
09	Search Terms based on the 3 criteria	**Normal** **Test Case 1** Course Name: Chess online regular course Student Name: Alex Status: Pending	**Test Case 1** Terms that match the search criteria will be displayed
		Abnormal **Test Case 2** Course Name: Robotics Course Student Name: Alex Status: Pending **Test Case 3** Course Name: Chess online regular course Student Name: October Status: Pending **Test Case 4**	**Test Case 2** No terms displayed as terms with course name “robotics course” don’t exist. **Test Case 3** No terms will be displayed as no term with student name “October” exists. **Test Case 4**

		Course Name: Chess online regular course Student Name: Alex Status: Closed	No terms will be displayed as no term with status “Closed” exists.
10	Edit Terms	**Normal** **Test Case 1** Sessions*: 8 End Date*: 14/02/2022 Discount Type: Sibling Discount Amount: 14 Payment Method: Bank Transfer Amount Paid: 86 Payment Date: 10/01/2022 Account Name: SmithPeter Status*: Paid	**Test Case 1** (changing status from pending to paid) A new term should be generated with the same details, and status pending
		Abnormal **Test Case 2** Sessions*: hello End Date*: 14/02/2022 Discount Type: Sibling Discount Amount: 14 Payment Method: Bank Transfer Amount Paid: 86 Payment Date: 10/01/2022 Account Name: SmithPeter Status*: Paid **Test Case 3** Refer to “Enter Term Details” section, as all the fields and checks are the same	**Test Case 2** (invalid sessions) Error message indicating invalid data entry and asking user to enter correct data **Test Case 3** Refer to “Enter Term Details” section, as all the fields and checks are the same
11		**Normal** **Test Case 1**	**Test Case 1**

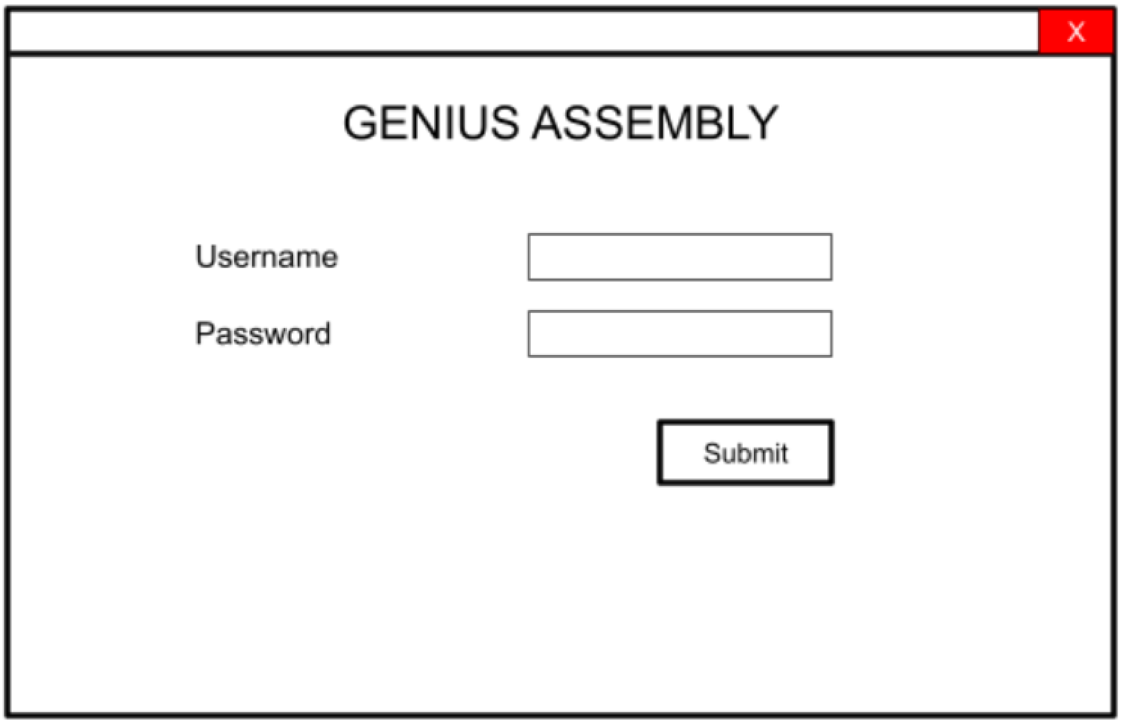
	Search Reports for Course	Search Criterion: Chess Online Regular Course	Course details for 'Chess Online Regular Course' displayed in the report section
		Abnormal **Test Case 2** Search Criterion: Robotics Course	**Test Case 2** No course displayed as no such course exists.
12	Search Reports for Schedule	**Normal** **Test Case 1** Search Criterion: Monday	**Test Case 1** Schedule details for 'Monday' displayed in the report section
		Abnormal **Test Case 2** Search Criterion: Sunday	**Test Case 2** No schedule displayed as no such schedule exists
13	Search Reports for Student	**Normal** **Test Case 1** Search Criterion: Alex	**Test Case 1** Students details for 'Alex' displayed in the report section
		Abnormal **Test Case 2** Search Criterion: October	**Test Case 2** No student displayed as no such student exists
14	Show Reports in Text File	**Test Case 1** Check each reports file to see whether the correct data is being stored in the correct file	**Test Case 1** Depending on which file the report is being saved, details for all relevant entities should be visible

<u>Initial Screen Designs</u>
User Authentication

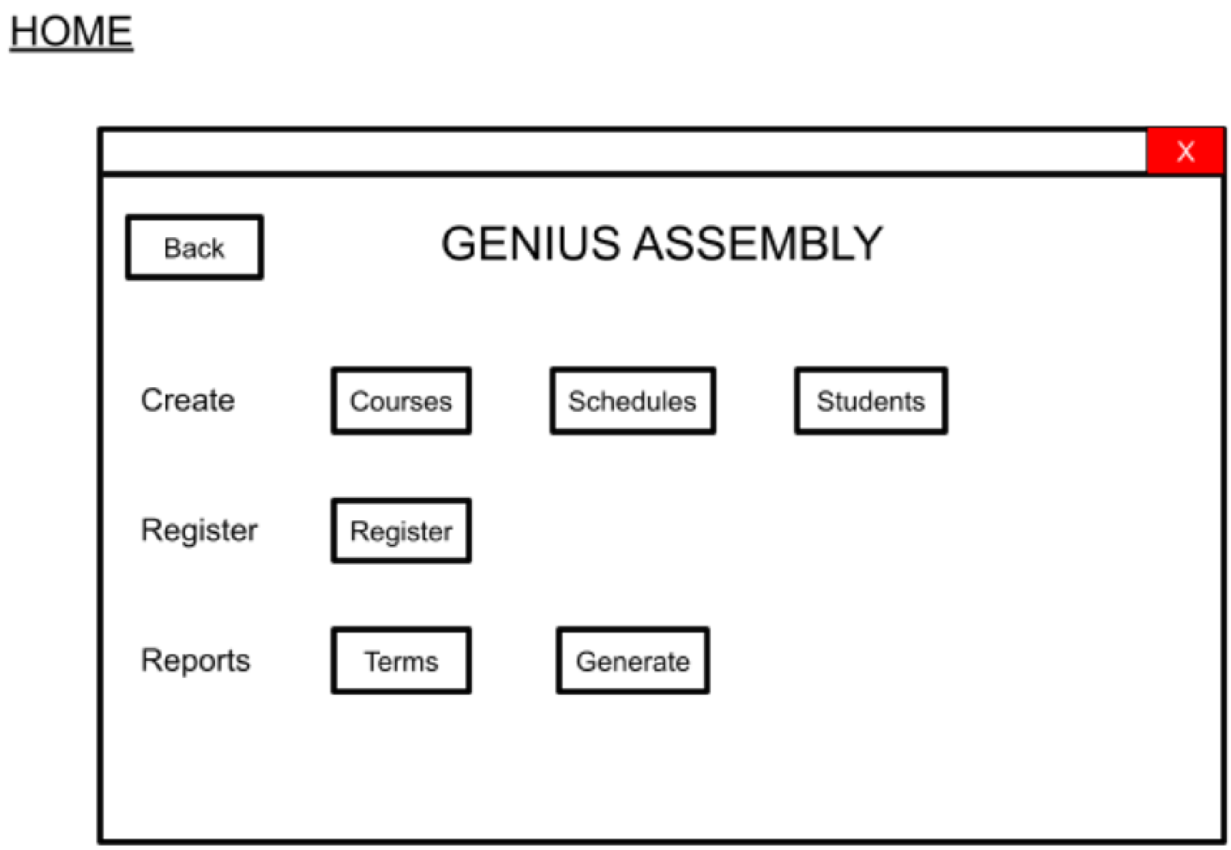
USER LOGIN



USER SIGN-UP

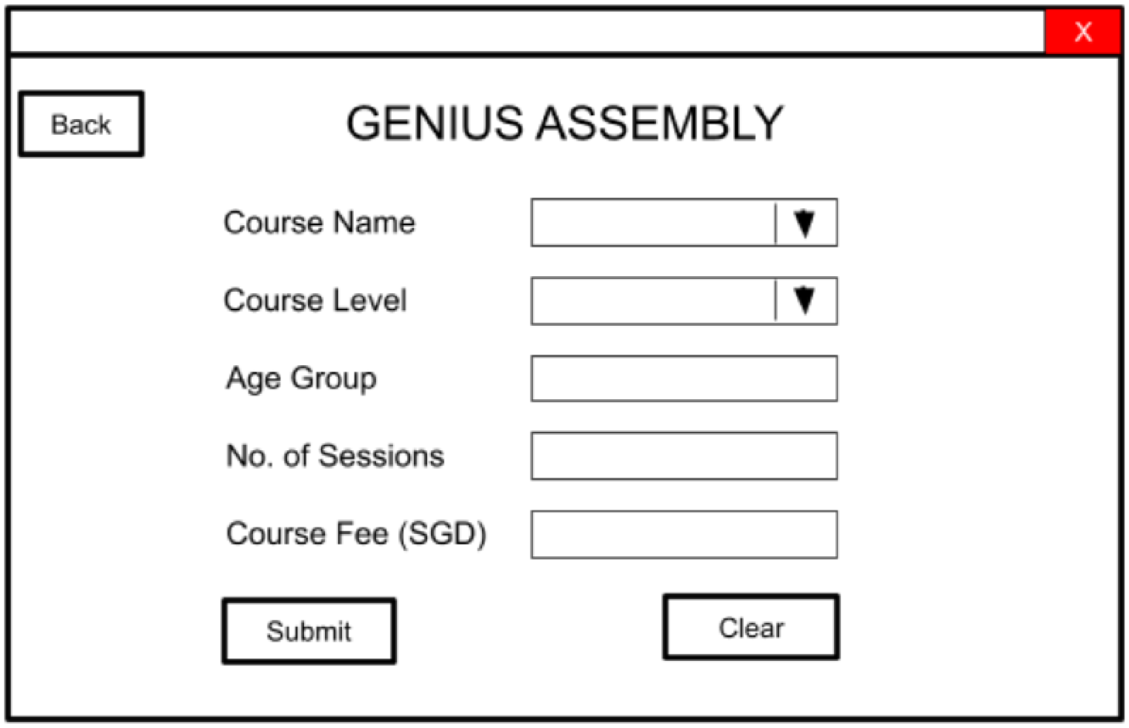


Home Page

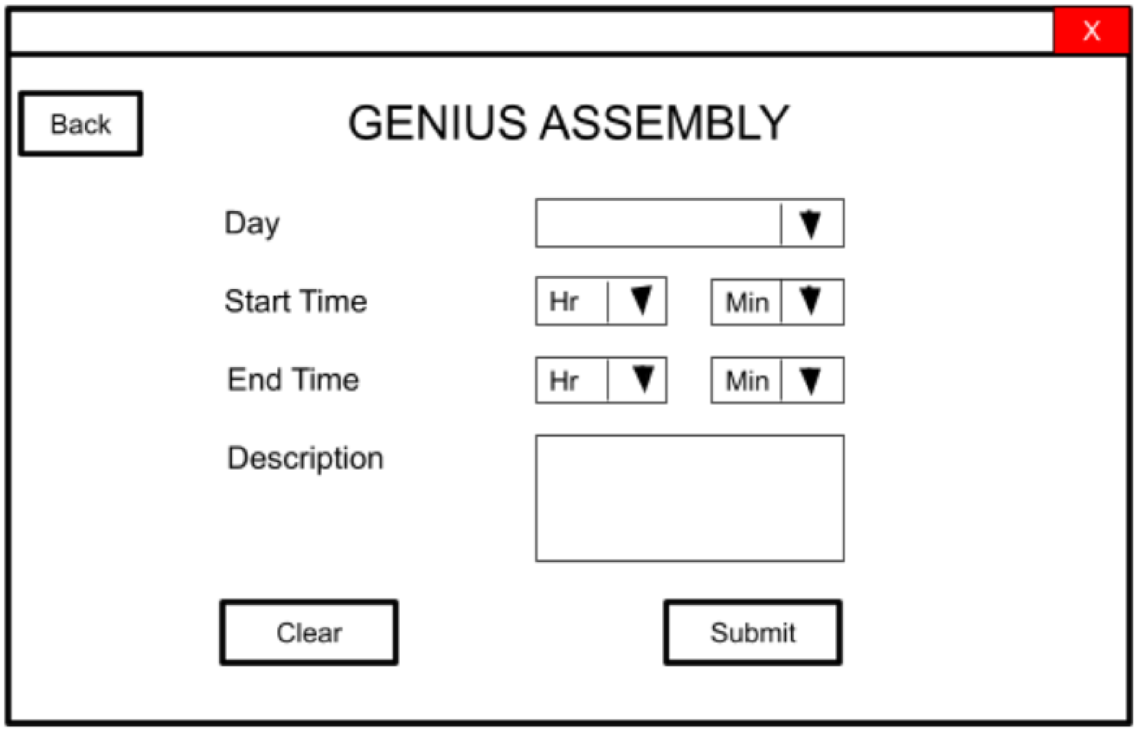

HOME
X
Back
GENIUS ASSEMBLY
Create
Courses
Schedules
Students
Register
Register
Reports
Terms
Generate

Creation of Course, Schedule and Student

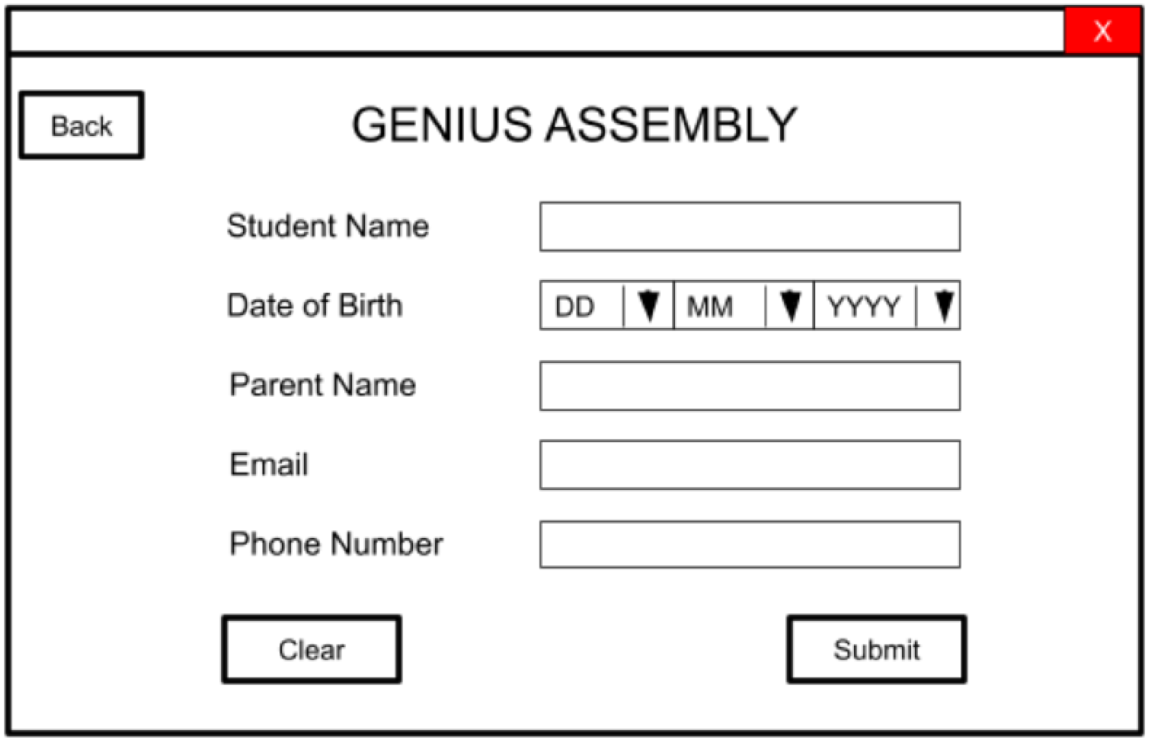
COURSE CREATION



SCHEDULE CREATION

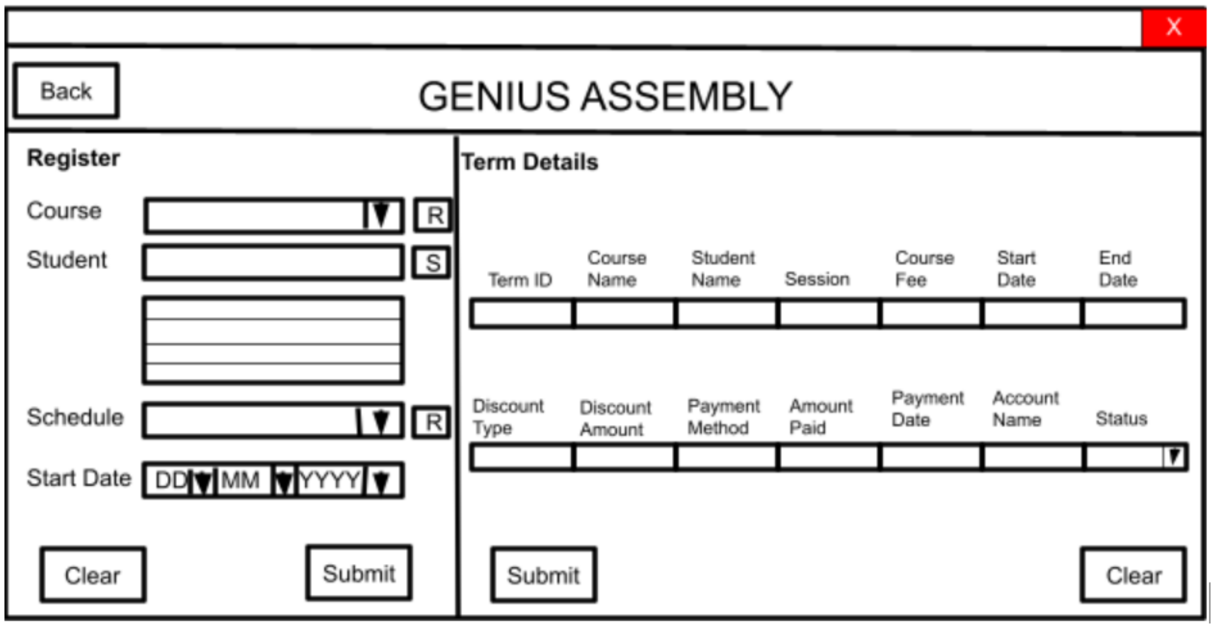


STUDENT CREATION

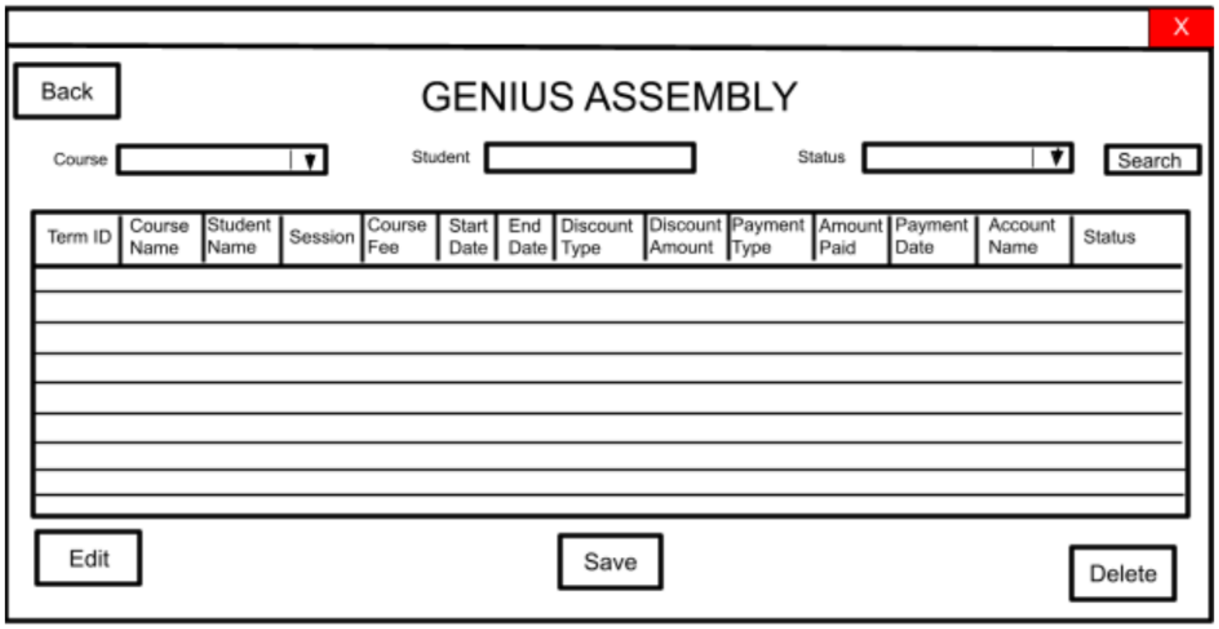


Enrollment

REGISTRATION

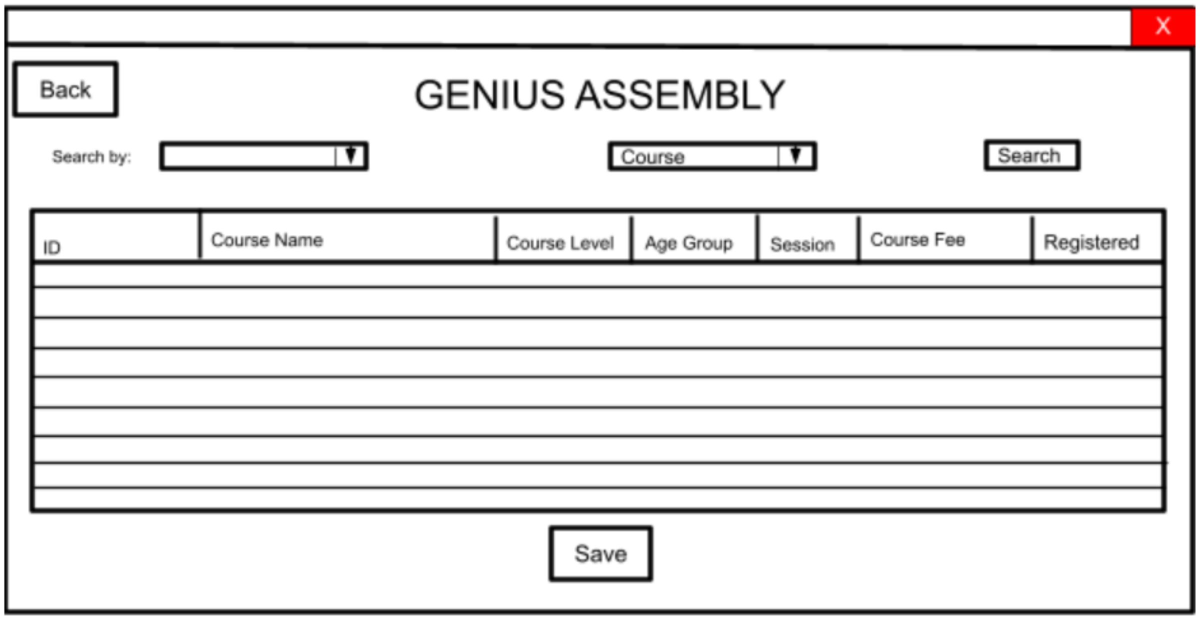


TERM DETAILS REPORT



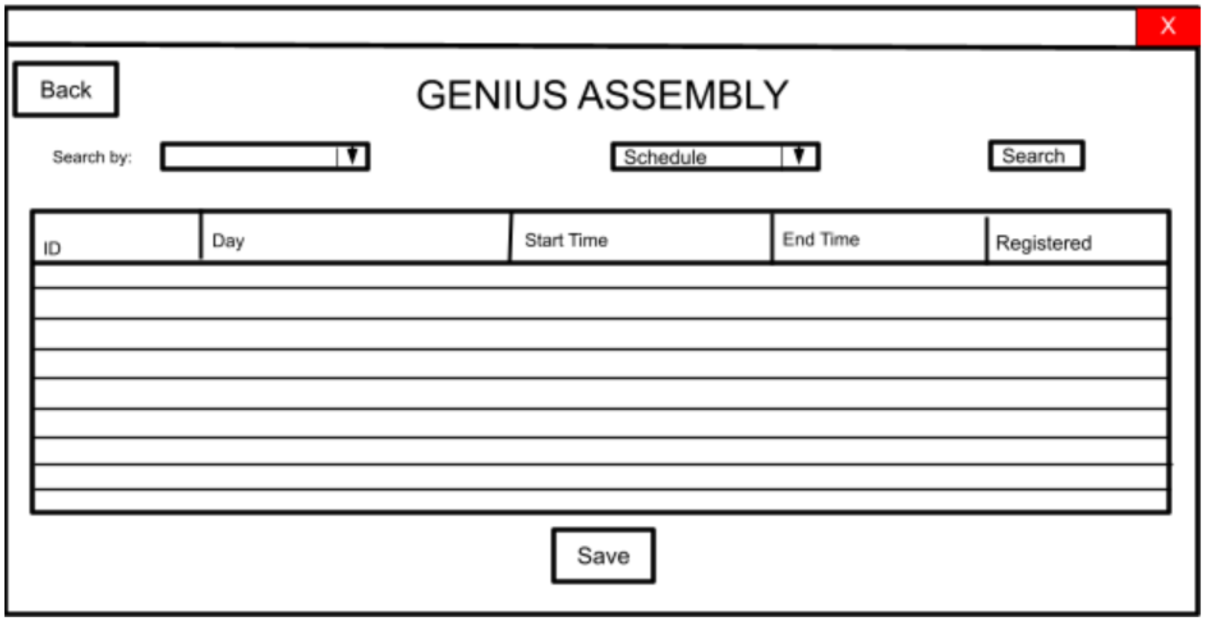
Reports

COURSE DETAILS REPORT



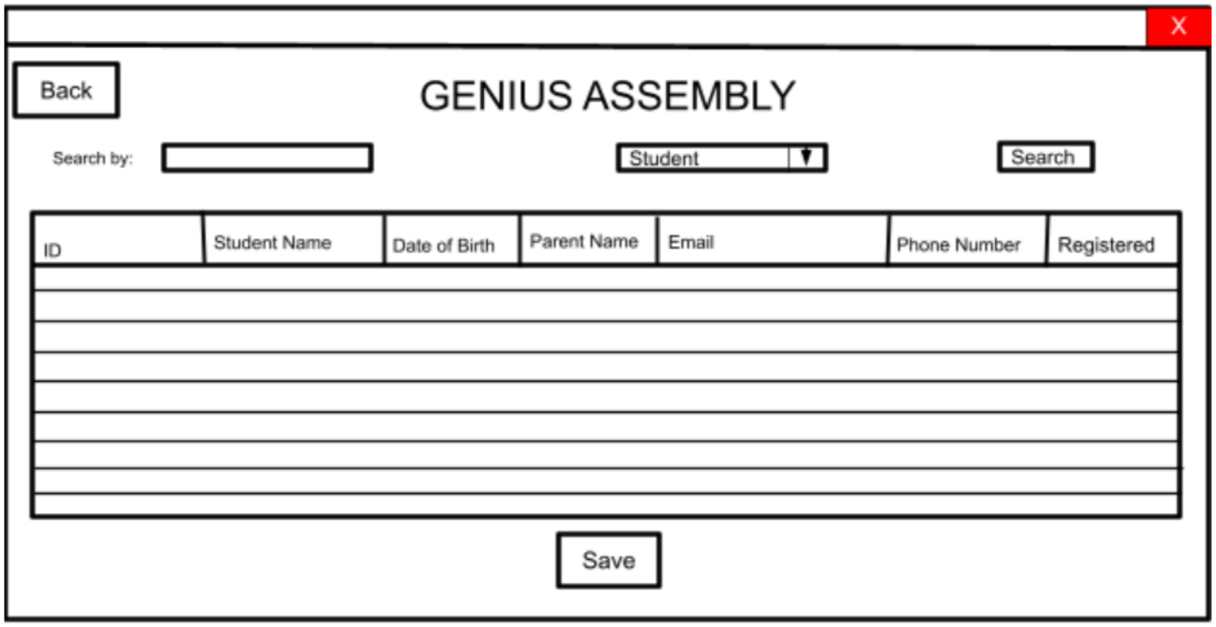
ID	Course Name	Course Level	Age Group	Session	Course Fee	Registered

SCHEDULE DETAILS REPORT



ID	Day	Start Time	End Time	Registered

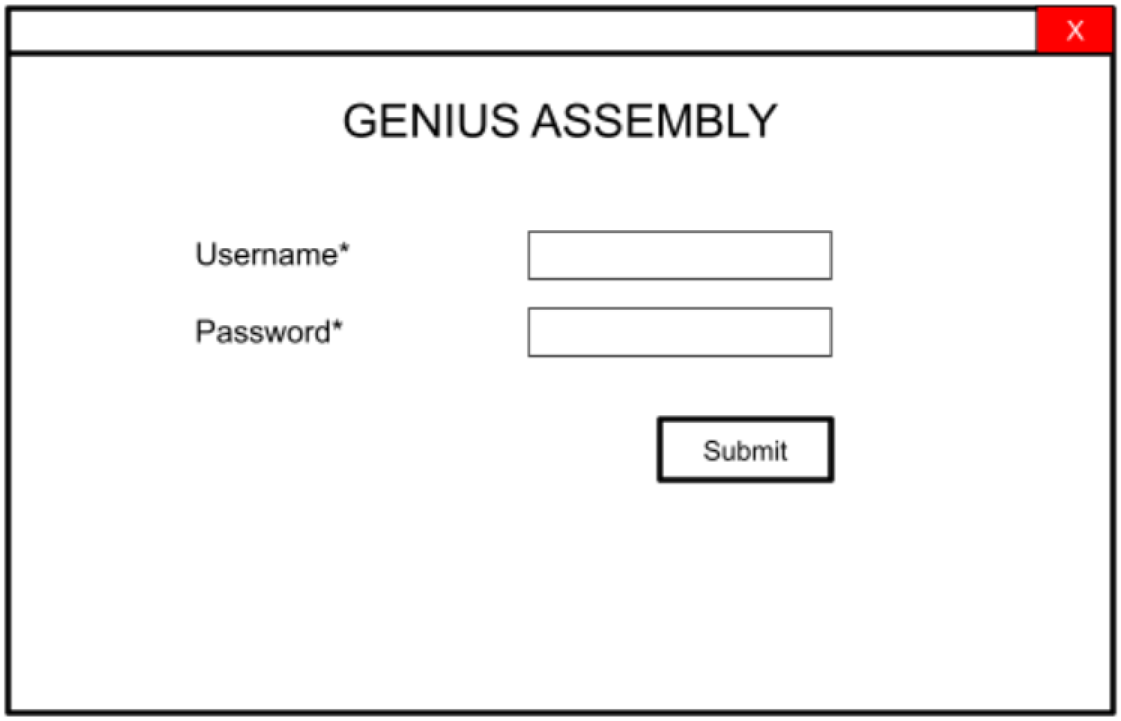
STUDENT DETAILS REPORT



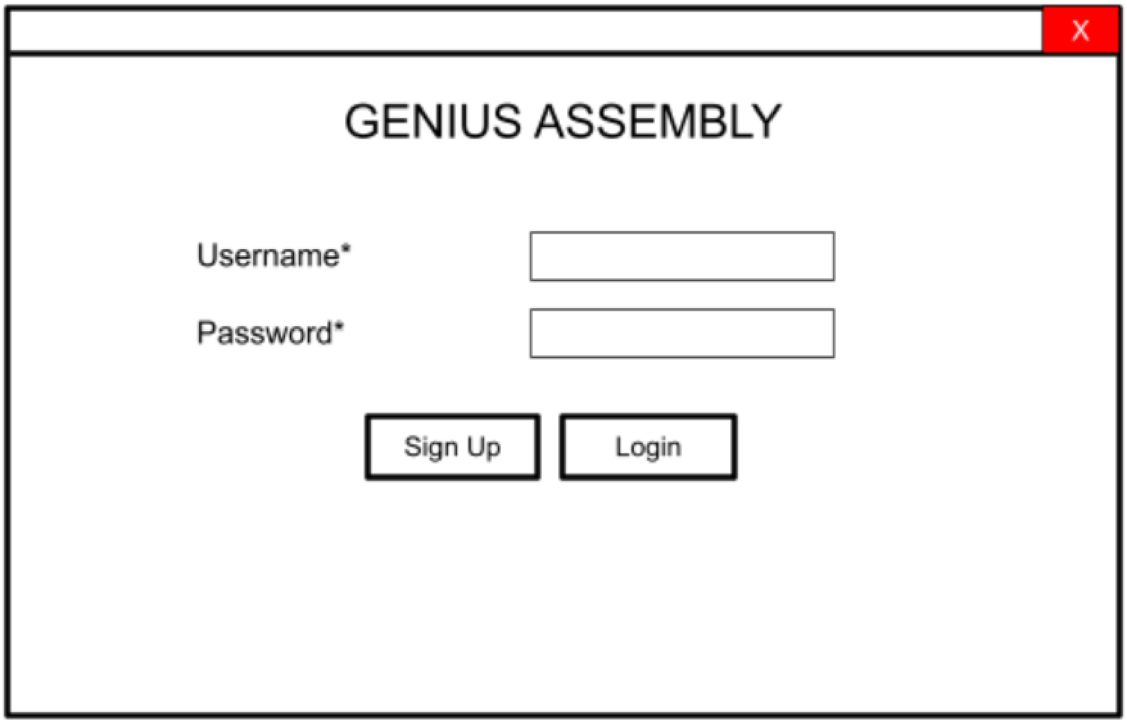
ID	Student Name	Date of Birth	Parent Name	Email	Phone Number	Registered

Final Screen Designs
User Authentication

User Sign Up

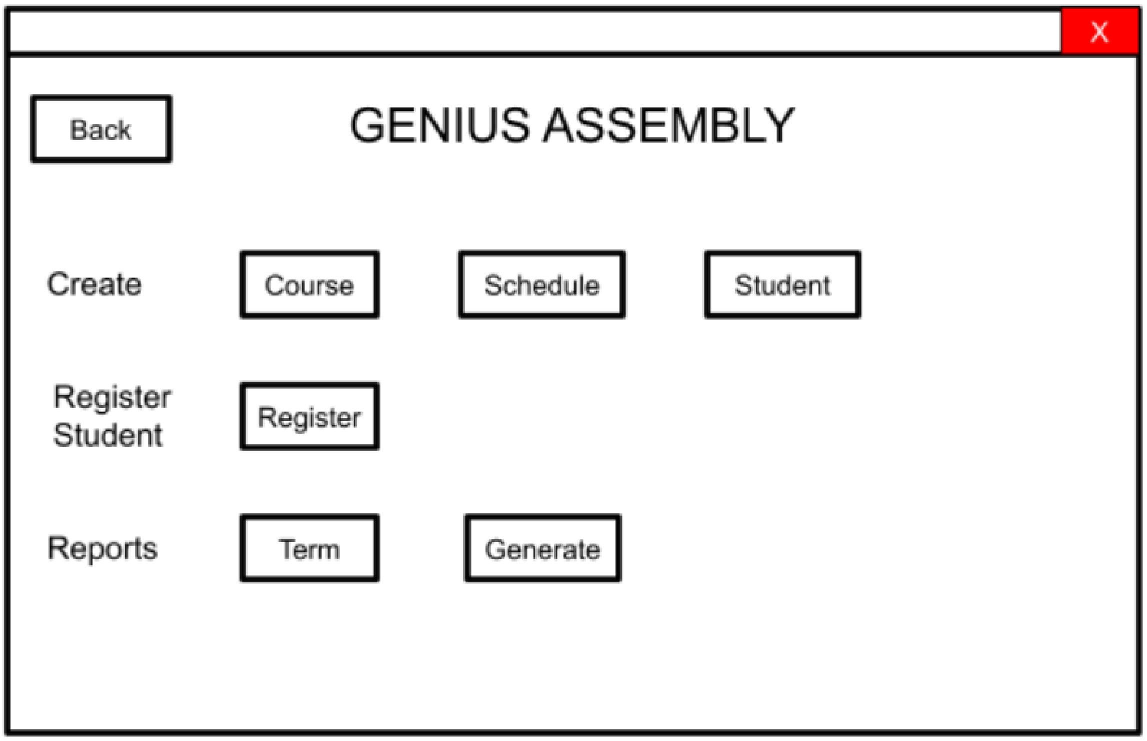


User Login



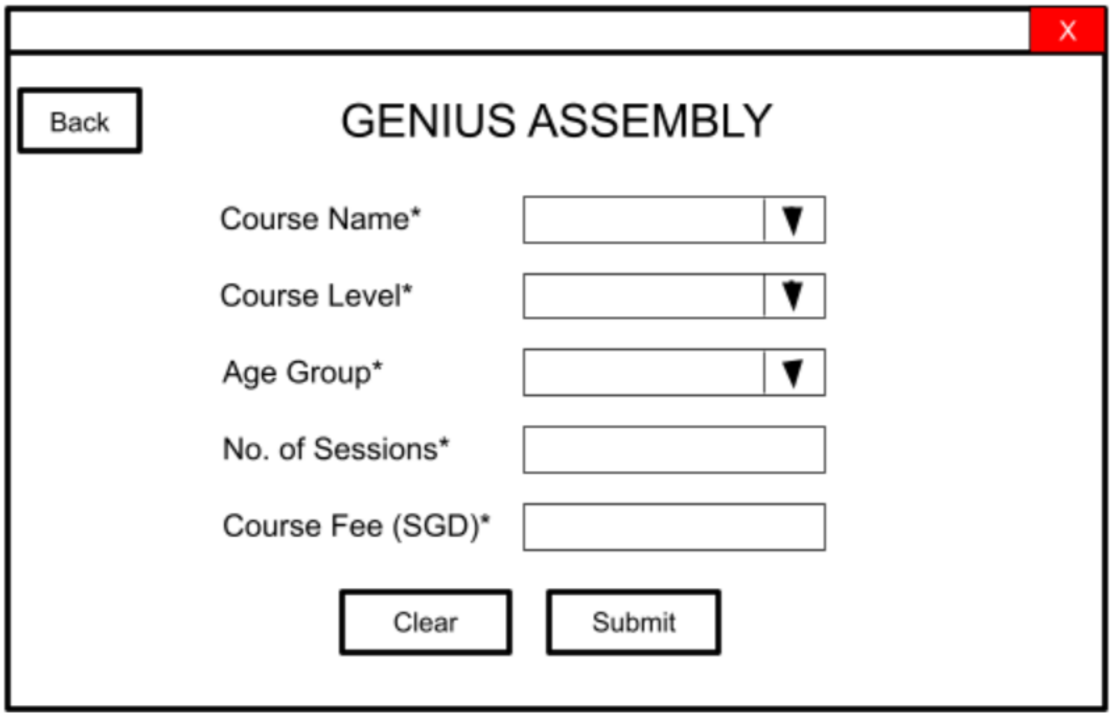
Home Page

Home Screen



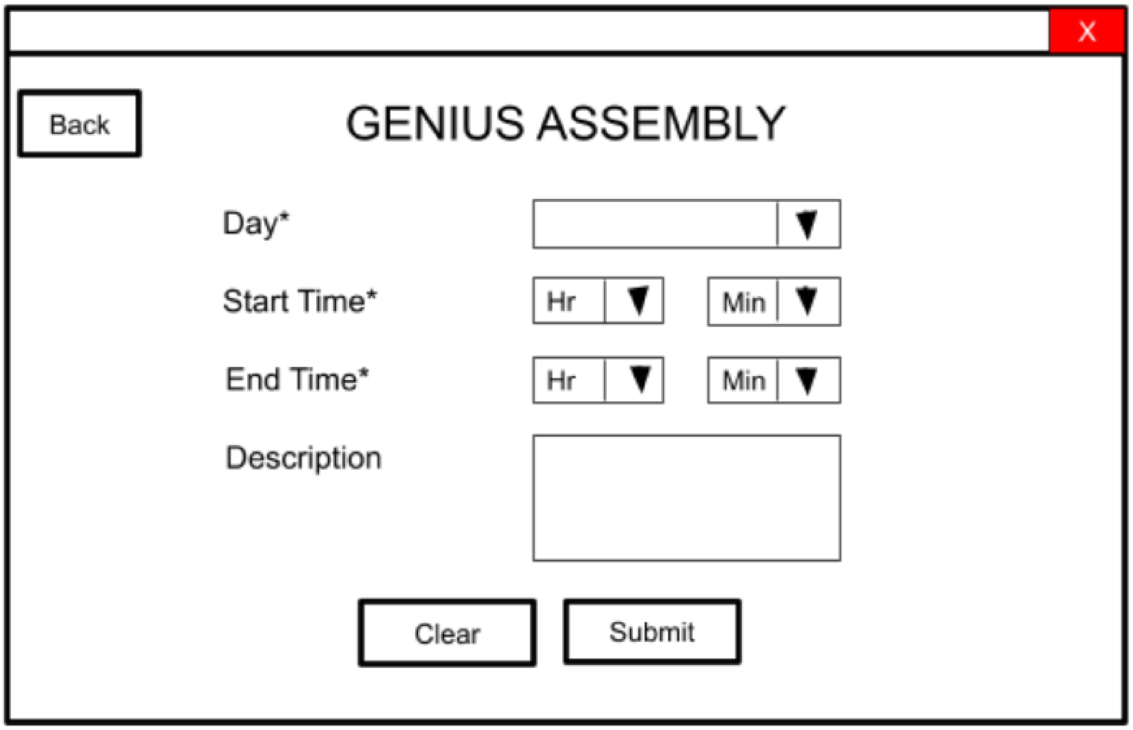
Creation of Course, Schedule and Student

Course Creation

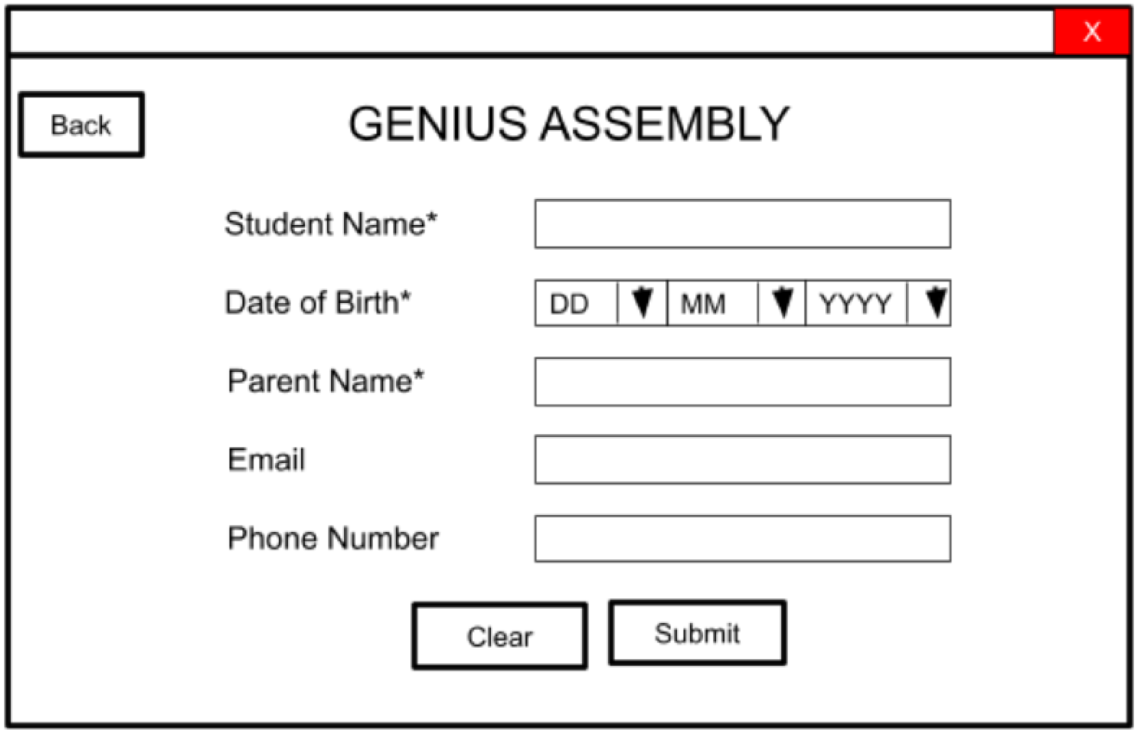


Course Name*
Course Level*
Age Group*
No. of Sessions*
Course Fee (SGD)*

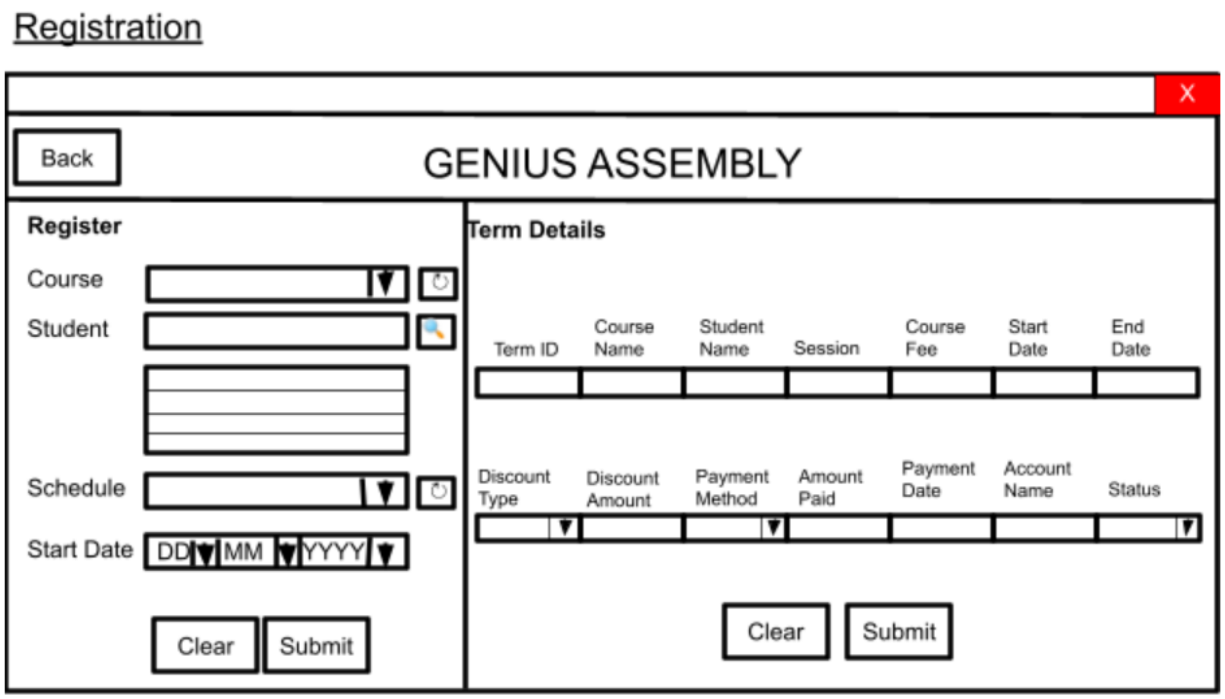
Schedule Creation



Student Creation

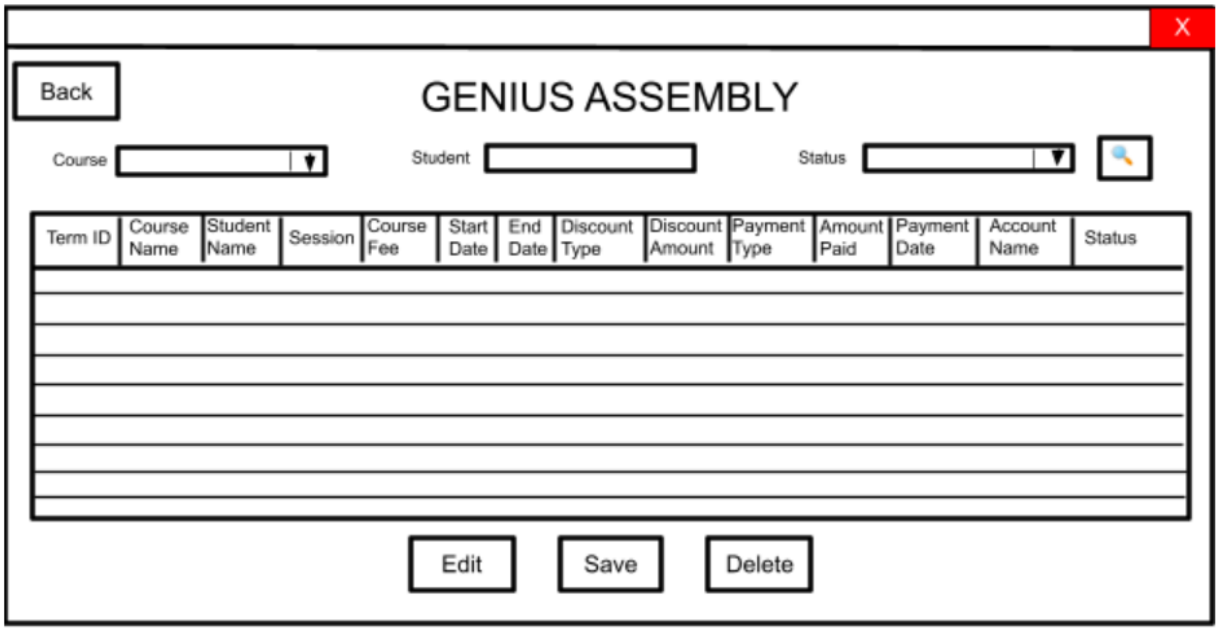


Enrollment


Registration
X
Back
GENIUS ASSEMBLY
Register
Course
Student
Schedule
Start Date
DD
MM
YYYY
Clear
Submit
Term Details
Term ID
Course Name
Student Name
Session
Course Fee
Start Date
End Date
Discount Type
Discount Amount
Payment Method
Amount Paid
Payment Date
Account Name
Status
Clear
Submit

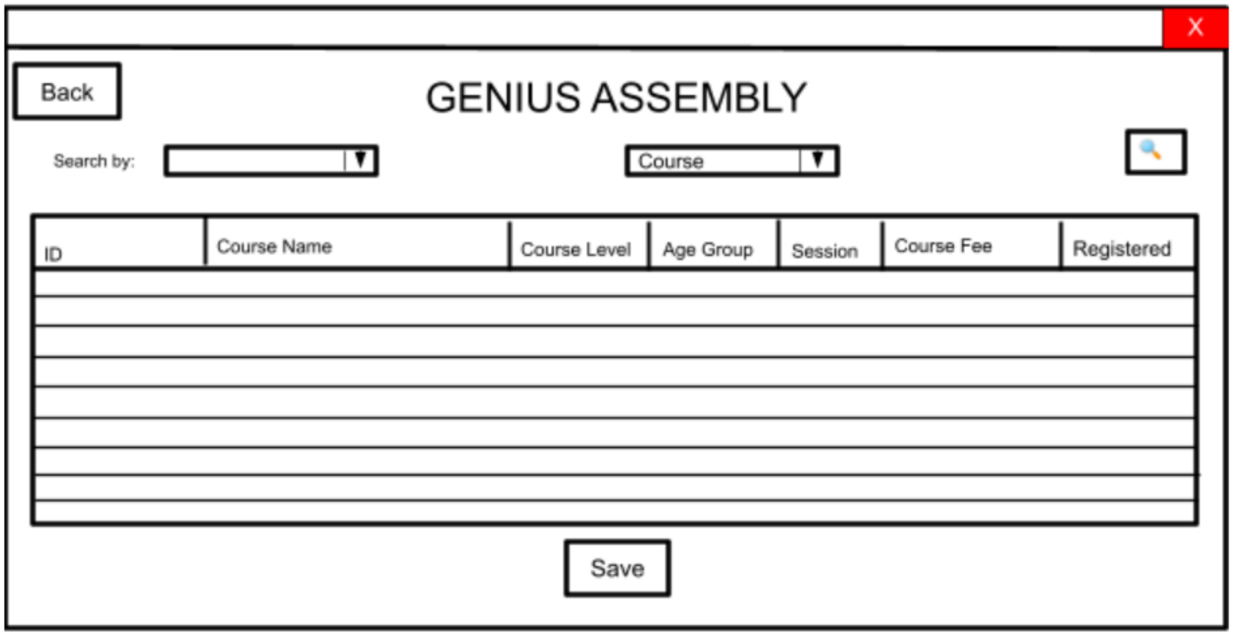
Reports

Term Details Report



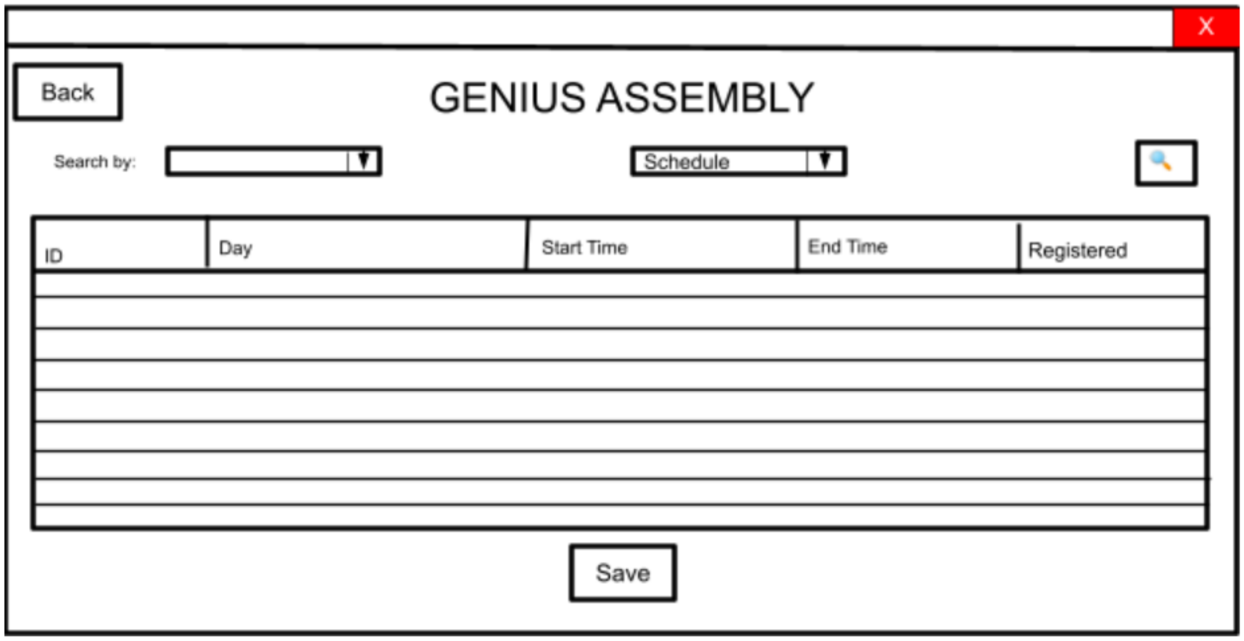
Term ID	Course Name	Student Name	Session	Course Fee	Start Date	End Date	Discount Type	Discount Amount	Payment Type	Amount Paid	Payment Date	Account Name	Status

Course Details Report

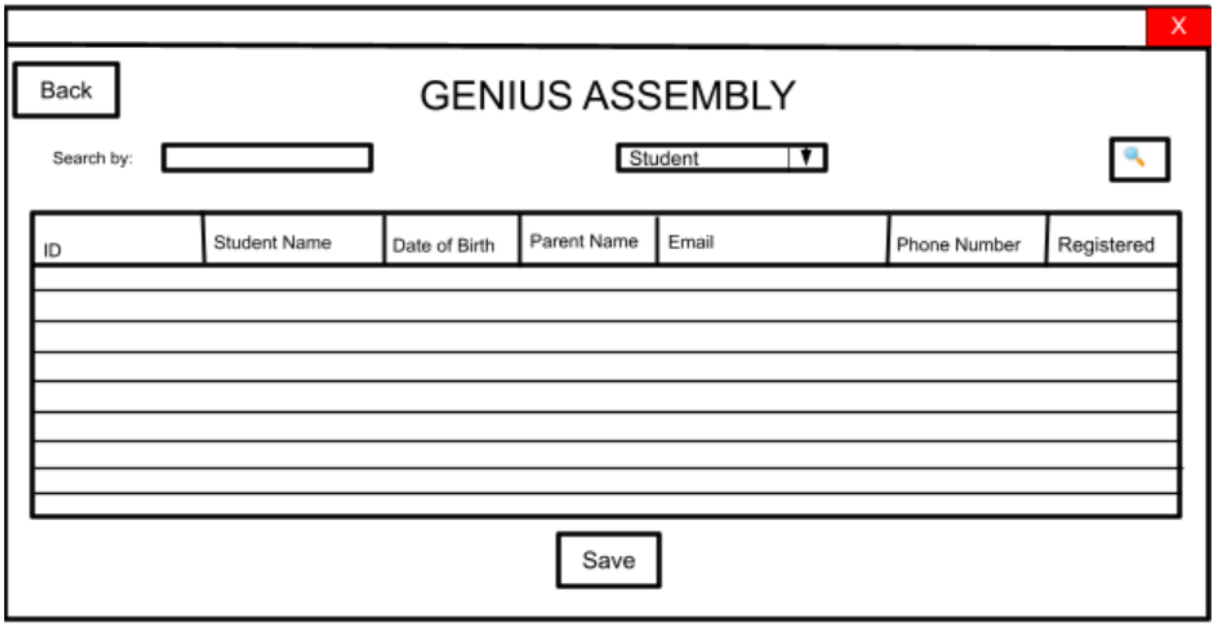


ID	Course Name	Course Level	Age Group	Session	Course Fee	Registered

Schedule Details Report



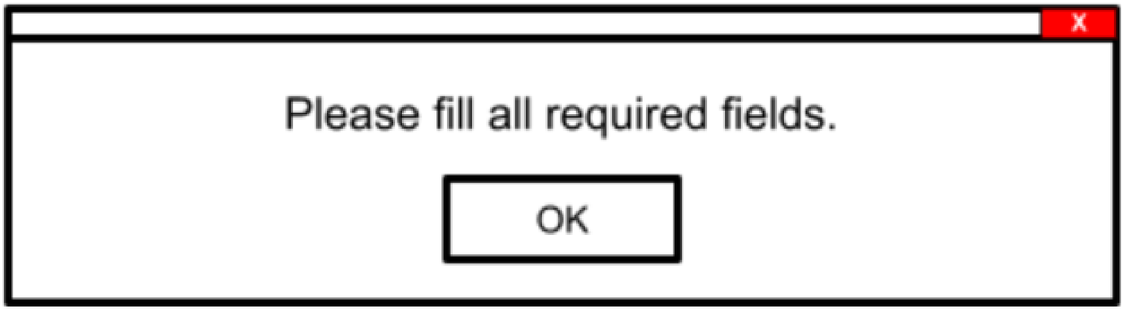
Student Details Report



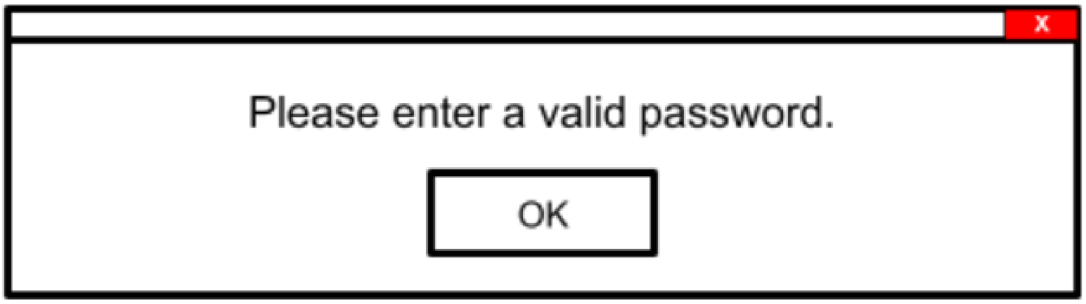
Common Functionality

Validation Errors

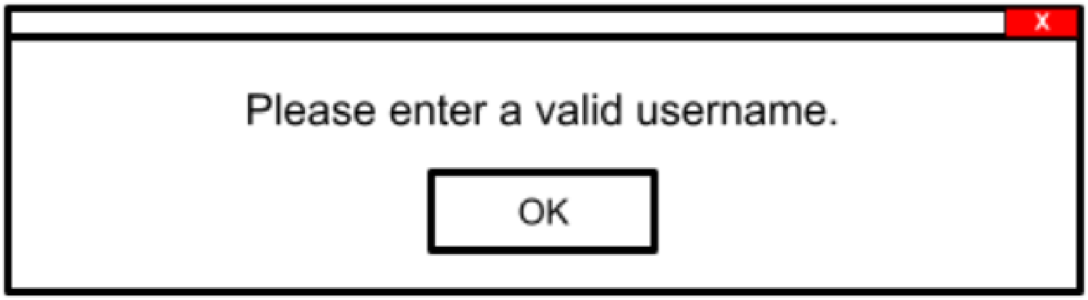
Unfilled Data


Please fill all required fields.

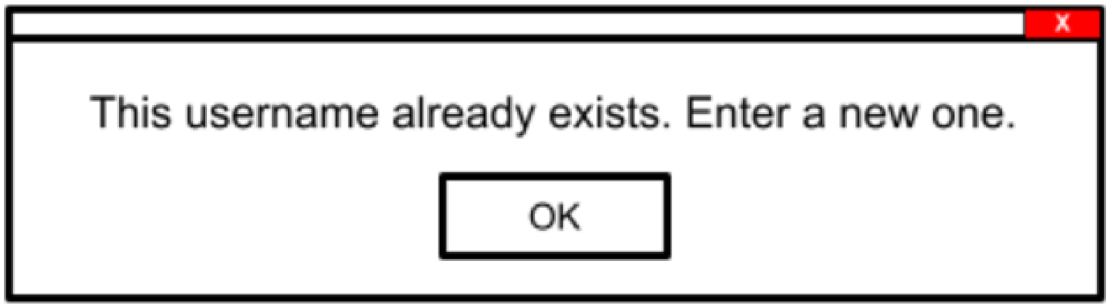
Passwords Do Not Match


Please enter a valid password.

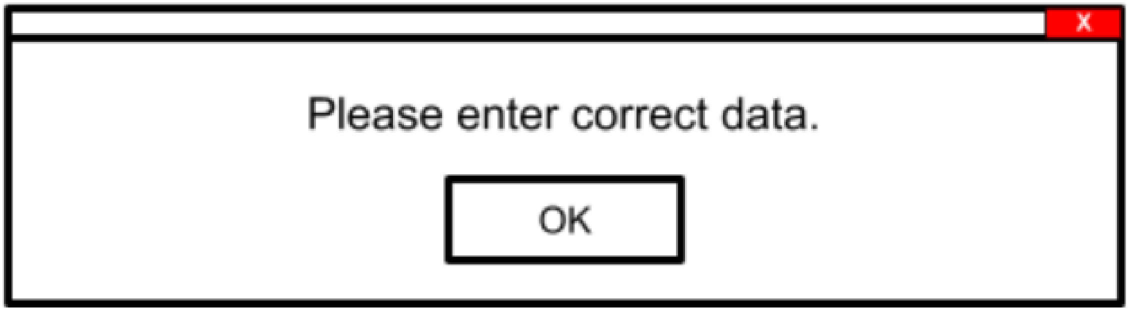
Username Not Found During Login


Please enter a valid username.

Username Clash During User Creation

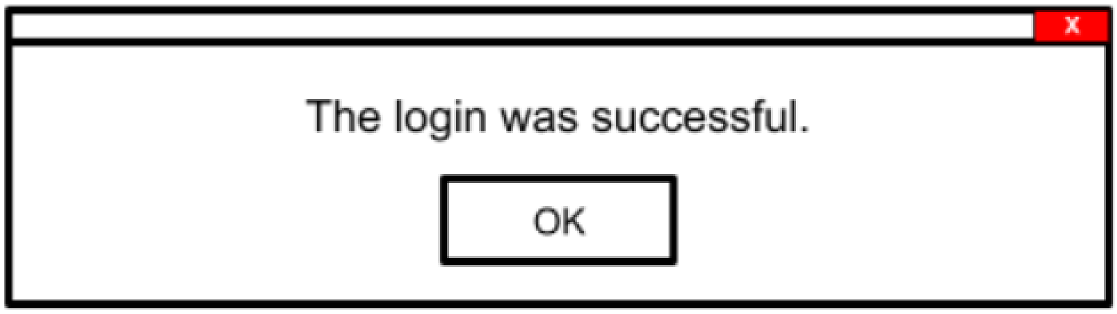

This username already exists. Enter a new one.

Invalid Data

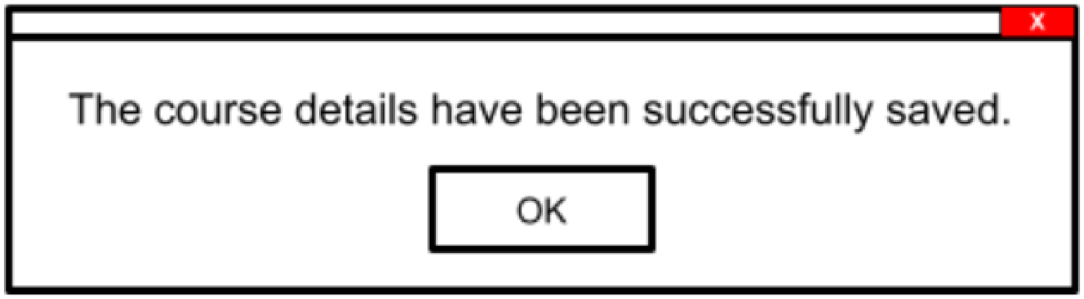

Please enter correct data.

Validation Successes

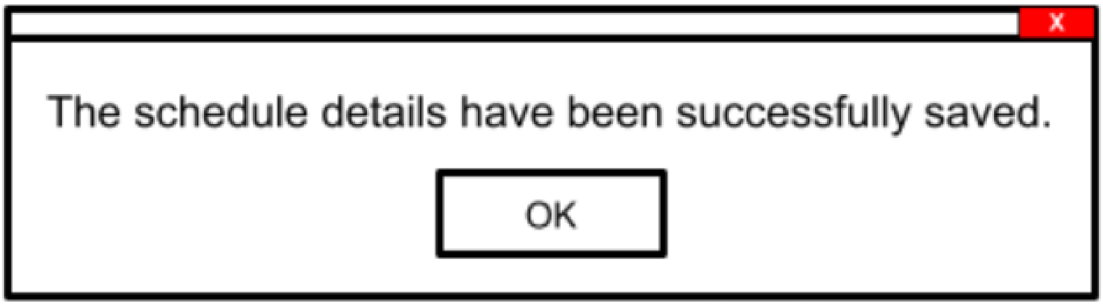
Successful Login


The login was successful.

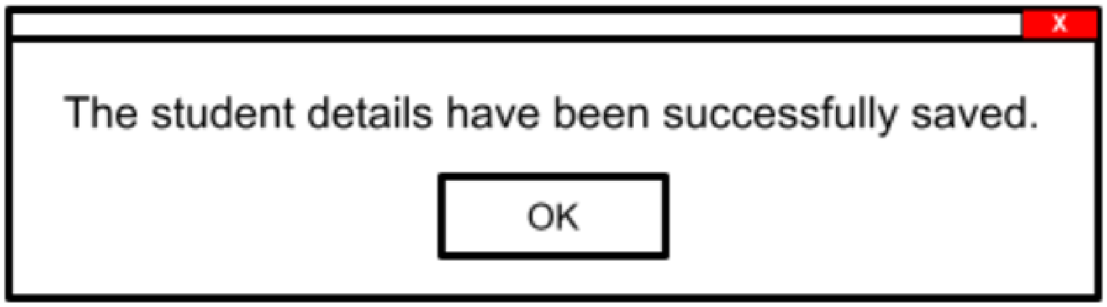
Successful Course Creation


The course details have been successfully saved.

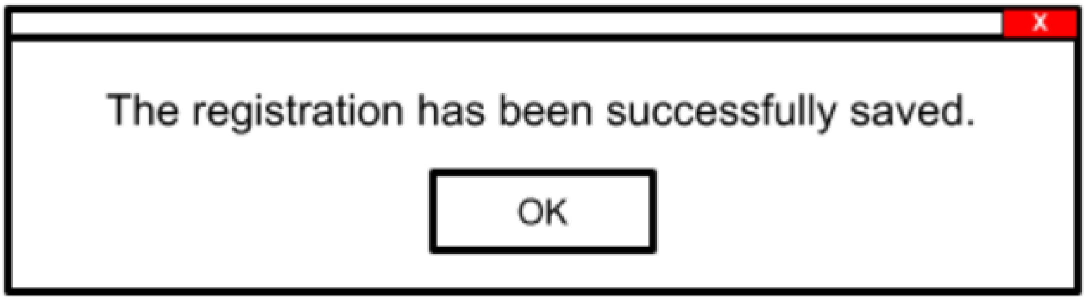
Successful Schedule Creation


The schedule details have been successfully saved.

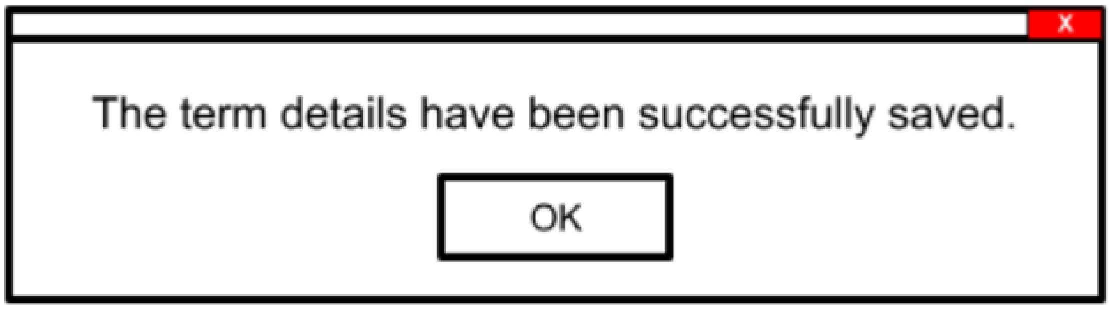
Successful Student Creation


The student details have been successfully saved.

Successful Registration

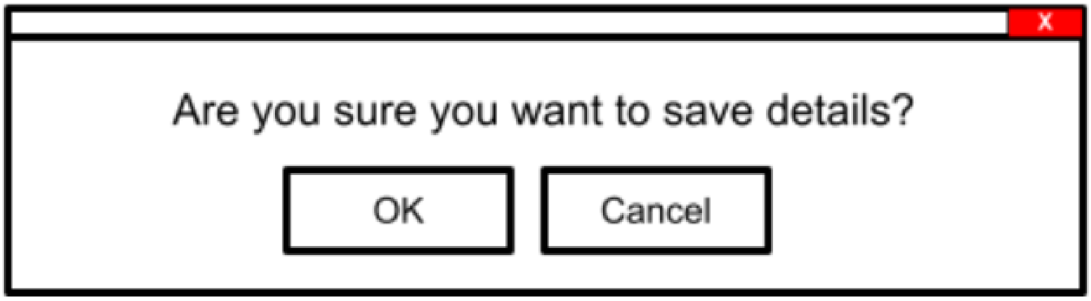

The registration has been successfully saved.

Successful Term Edit

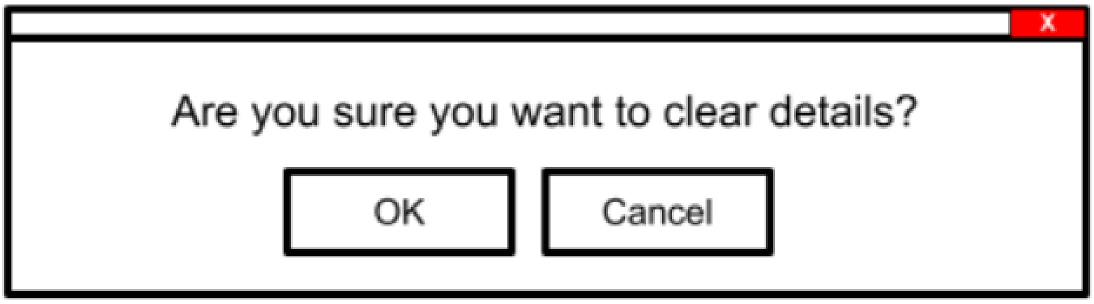

The term details have been successfully saved.

Verification Messages

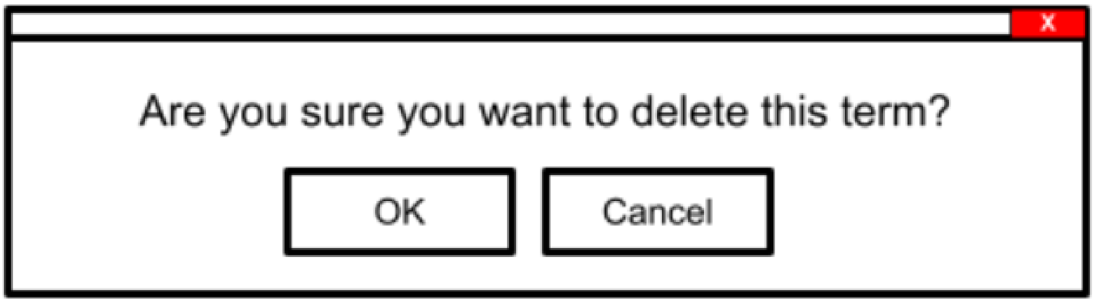
Submit Details


Are you sure you want to save details?

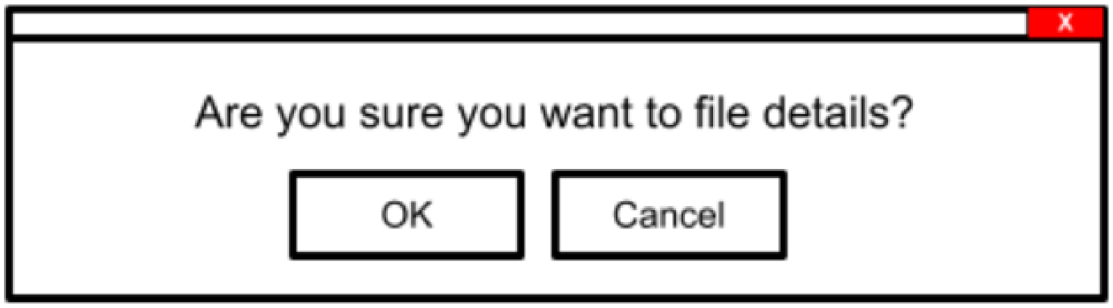
Clear Details


Are you sure you want to clear details?

Delete Terms


Are you sure you want to delete this term?

File Reports


Are you sure you want to file details?

Pseudocode

```
BINARY SEARCH

// Function to search through an array
procedure binarySearch(ARRAY, LOW, HIGH, TARGET)

// ARRAY is the sorted array of elements, TARGET is the value to be found
// initially, LOW is index position 0 and HIGH is number of array elements - 1

        if HIGH >= LOW then
                MID = (HIGH + LOW) div 2        // MID is the average of HIGH and LOW indices
                if ARRAY[MID] = TARGET then
                        return true     // The value has been found
                else if ARRAY[MID] > TARGET then
                        // Recursively repeat binary search on the left side of MID
                        return binarySearch(ARRAY, LOW, MID - 1, TARGET)
                else
                        // Recursively repeat binary search on the right side of MID
                        return binarySearch(ARRAY, MID + 1, HIGH, TARGET)
                end if
        else
                return false    // The value has not been found
        end if
end procedure

-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
```

```
QUICK SORT

// Function that determines the pivot index
procedure partition(LOW, HIGH, ARRAY)
        I = LOW - 1     // Index of the smaller element
        // Shows the correct position of PIVOT so far
        TEMP = 0
        PIVOT = ARRAY[HIGH]     // Selecting the last element as PIVOT
                                // This element will be placed in its correct position
        loop J from LOW to HIGH
                if ARRAY[J] <= PIVOT then       // If the current element is smaller than the
                                                   PIVOT
                        I = I + 1               // Increment the index of the smaller element
                        TEMP = ARRAY[I]         // and swap the 2 elements
                        ARRAY[I] = ARRAY[J]
                        ARRAY[J] = TEMP
                end if
        end loop

        // At this point, all elements smaller than PIVOT are on the left of where the PIVOT
           should be
        // All elements greater than PIVOT are on the right of where PIVOT should be
        TEMP = ARRAY[I + 1]             // Placing PIVOT in its correct position
        ARRAY[I + 1] = ARRAY[HIGH]
        ARRAY[HIGH] = TEMP

        return I + 1    // Return index position of PIVOT
end procedure
```

```
// Function to perform quicksort
procedure quickSort(LOW, HIGH, ARRAY)
        if LOW < HIGH then
                MID = partition(LOW, HIGH, ARRAY)       // Determining the PIVOT
                quickSort(LOW, MID - 1, ARRAY)          // Recursively calling quicksort on
                                                           the left side of PIVOT

                quickSort(MID - 1, HIGH, ARRAY)         // Recursively calling quicksort on
                                                           the right side of PIVOT
        end if
end procedure
```


ENCRYPTION AND QUEUE(CEASAR CIPHER)

```
// Function that encrypts plain text and returns cipher text
procedure encrypt(STRING, SHIFT)
        ENCRYPTED_STRING = ''   // Initialising final encrypted string (cipher text)
        QUEUE = new Queue()
        loop for each CHARACTER in STRING       // Looping through each character in STRING
                ASCII_CODE = ASCII value of CHARACTER   // Converting each character to
                                                           numeric ASCII value
                ENCRYPTED_ASCII_VALUE = ASCII_CODE + SHIFT      // Encryption Key

                // Converting ASCII value back to corresponding character
                ENCRYPTED_CHARACTER = CHARACTER corresponding to ENCRYPTED_ASCII_VALUE
                QUEUE.enqueue(ENCRYPTED_CHARACTER)      // Adding each character in STRING
                                                           to QUEUE
        end loop

        loop while NOT(QUEUE.isEmpty()) // Condition for underflow
                ENCRYPTED_CHARACTER = QUEUE.dequeue()   // Removing character from QUEUE
                // Concatenating to form cipher text
                ENCRYPTED_STRING = ENCRYPTED_STRING + ENCRYPTED_CHARACTER
        end loop

        return ENCRYPTED_STRING
end procedure

// Function that decrypts cipher text and returns plain text
procedure decrypt(STRING, SHIFT)
        DECRYPTED_STRING = ''   // Initialising final decrypted string (plain text)
        QUEUE = new Queue()
        loop for each CHARACTER in STRING       // Looping through each character in STRING
                ASCII_CODE = ASCII value of CHARACTER   // Converting each character to
                                                           numeric ASCII value
                DECRYPTED_ASCII_VALUE = ASCII_CODE + SHIFT      // Decryption Key


                // Converting ASCII value back to corresponding character
                DECRYPTED_CHARACTER = CHARACTER corresponding to DECRYPTED_ASCII_VALUE
                QUEUE.enqueue(DECRYPTED_CHARACTER)      // Adding each character in STRING
                                                           to QUEUE
        end loop
        loop while NOT(QUEUE.isEmpty()) // Condition for underflow
                DECRYPTED_CHARACTER = QUEUE.dequeue()   // Removing character from QUEUE
                // Concatenating to form plain text
                DECRYPTED_STRING = DECRYPTED_STRING + DECRYPTED_CHARACTER
        end loop

        return DECRYPTED_STRING
end procedure

---------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------

STACK

HOME_FRAME = newFrame() // newFrame() returns a GUI Frame. This function has been used to
                           initialise the homepage

STACK = newStack()      // Initialising an empty stack
STACK.push(HOME_FRAME)  // Adding the homepage to the stack

// The top element of the STACK displays the frame the user is currently on

// Function that deletes a frame from the stack
// Called when the user wants to access a new GUI Frame
procedure add_element(FRAME)
        STACK.push(FRAME)       // Adding desired FRAME to the STACK
        FRAME.display() // FRAME.display() displays the frame (passed as a parameter) to
                           the user and hides all other frames
end procedure


// Function that deletes a frame from the stack
// Called when the BACK button is pressed
procedure delete_element()
        STACK_LENGTH = STACK.getLength()        // STACK.getLength() returns the number
                                                   of elements present in the stack
        if STACK_LENGTH > 1 then        // Condition to ensure that there is no
                                           underflow. There will always be one element
                                           present in the stack
                                        // This element is the homepage frame. This will
                                           never be deleted from the stack.
                STACK.pop()                     // Removing the last frame present in the
                                                   stack
                PREVIOUS_FRAME = STACK.peek()   // STACK.peek() returns the frame present
                                                   at the top of the stack without
                                                   deleting it from the stack

                PREVIOUS_FRAME.display()        // PREVIOUS_FRAME.display() displays the
                                                   frame (passed as a parameter) to the
                                                   user and hides all other frames
        end if
end procedure

---------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
```

```
COURSE CREATION

// Initialising class to make course object
MAKE_COURSE = newClass()

// Function that creates a course based on user-input
procedure create_course(COURSE_NAME, COURSE_LEVEL, AGE_GROUP, FEES, DURATION)
        // User is allowed to enter data for 5 different fields that will be used to make
           a particular course
        // All of these are required fields
        // Checking if either of the required fields are blank
        // The user can only create a course if all required fields are filled
        if COURSE_NAME = "" or
           COURSE_LEVEL = "" or
           AGE_GROUP = "" or
           FEES = 0 or
           DURATION = 0 then
                output "Error. Fill all required fields."       // Some fields have been
                                                                   left blank
        else
                // All fields have been filled
                // Session and Fees need to be validated to ensure that they follow the
                   correct format
                if FEES.validate() and DURATION.validate() then // FEES.validate() and
                                                                   DURATION.validate() are
                                                                   user_defined functions
                                                                // They return a boolean
                                                                   value
                                                                // Depending on the
                                                                   parameter, they carry
                                                                   out the required
                                                                   validation checks

                        // Generating a unique identifier for each course made. This will
                           be done in 3 steps

                        // COURSE_NAMES_ARRAY is a pre-defined array containing acceptable
                           course names
                        // It stores all course_names that can be used in the course
                           making process
                        // The .indexPosition(COURSE_NAME) function returns the index
                           position of COURSE_NAME in the array COURSE_NAMES_ARRAY
                        PART_A = COURSE_NAMES_ARRAY.indexPosition(COURSE_NAME)

                        // asciiValue() returns the equivalent ASCII value for that
                           character
                        // The value returned is of type integer
                        // COURSE_LEVEL.firstCharacter() returns the first character of
                           the string COURSE_LEVEL
                        PART_B = asciiValue(COURSE_LEVEL.firstCharacter())

                        // AGE_GROUPS_ARRAY is a pre-defined array containing acceptable
                           age_groups
                        // It stores all course_names that can be used in the course making
                           process
                        // The .indexPosition(COURSE_NAME) function returns the index
                           position of COURSE_NAME in the array COURSE_NAMES_ARRAY
                        PART_C = AGE_GROUPS_ARRAY.indexPosition(AGE_GROUPS)
                        // Concatenating the 3 parts to obtain the final unique ID
                        // The string() function returns the value passed into it as a string
                        COURSE_ID = string(PART_A) + string(PART_B) + string(PART_C)

                        // Creating a new course based on user input
                        // A course is created using OOP concepts, where each of the fields
                           entered are attributes of the course object, including the
                           course_id
                        CREATED_COURSE = MAKE_COURSE()

                        // writeFile("course_file.dat", CREATED_COURSE) writes CREATED_COURSE
                           to the file named "course_file.dat"
                        writeFile("course_file.dat", CREATED_COURSE)

                else
                        // Fees or Duration validation failed. User must reenter the values
                           present in the field(s).
                        output "Error. Enter correct values for duration and fees."
                end if
        end if
end procedure

*all .validate() procedures return True if the validation is successful and no error is found
---------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
```

```
SCHEDULE CREATION

// Initialising class to make schedule object
MAKE_SCHEDULE = newClass()

// Function that creates a schedule based on user-input
procedure create_schedule(DAY, START_TIME, END_TIME, DESCRIPTION)
        // User is allowed to enter data for 5 different fields that will be used to make a
           particular schedule
        // DAY, START_TIME AND END_TIME are the only required fields
        // Checking if either of the required fields are blank
        // The user can only create a schedule if all required fields are filled
        if DAY = "" or
           START_TIME = 0 or
           END_TIME = 0 or then
                output "Error. Fill all required fields."       // Some fields have been
                                                                   left blank
        else
                // All fields have been filled
                // The Start_time and End_time fields need to be validated to ensure that
                   they follow the correct format
                if START_TIME < END_TIME then   // If the user specifies that the start
                                                   time is greater than the end time, it
                                                   means that the schedule ends before it
                                                   starts. This is not possible, and hence
                                                   this check must be made

                        // Generating a unique identifier for each schedule made. This will
                           be done in 3 steps
                        // DAYS_ARRAY is a pre-defined array containing days from Monday to
                           Sunday
```

```
                    // The .indexPosition(DAY) function returns the index position of
                       DAY in the array DAYS_ARRAY
                    PART_A = DAYS_ARRAY.indexPosition(DAY)

                    // The Start time consists of Hours and Minutes. Hence only the
                       hours value is considered
                    PART_B = START_HOUR

                    // FILE_CONTENTS.getLength() returns the number of student objects
                       present in the file
                    PART_C = FILE_CONTENTS.getLength() + 1

                    // Concatenating the 3 parts to obtain the final unique ID
                    // The string() function returns the value passed into it as a type
                       string
                    SCHEDULE_ID = string(PART_A) + string(PART_B) + string(PART_C)

                    // Creating a new schedule based on user input
                    // A schedule is created using OOP concepts, where each of the fields
                       entered are attributes of the schedule object, including the
                       schedule_id
                    CREATED_SCHEDULE = MAKE_SCHEDULE()

                    // writeFile("schedule_file.dat", CREATED_SCHEDULE) writes
                       CREATED_SCHEDULE to the file named "schedule_file.dat"
                    writeFile("schedule_file.dat", CREATED_SCHEDULE)

                else
                    // The validation of timing of schedules failed. User must reenter
                       the values present in the field(s).
                    output "Error. Enter correct values for start and end time."
                end if
        end if
end procedure


*all .validate() procedures return True if the validation is successful and no error is found
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------

STUDENT CREATION

// Initialising class to make student object
MAKE_STUDENT = newClass()

// Function that makes a student based on user-input
procedure create_student(STUDENT_NAME, DATE_OF_BIRTH, PARENT_NAME, EMAIL, PHONE_NUMBER)
        // User is allowed to enter data for 5 different fields that will be used to make a
           particular student
        // Checking if either of the required fields are blank
        // The user can only create a student if all fields are filled
        if STUDENT_NAME = "" or
           DATE_OF_BIRTH = "" or
           PARENT_NAME = "" then
                output "Error. Fill all required fields."       // Some fields have been left
                                                                   blank
        else
```

```
                // All fields have been filled
                // Session and Fees need to be validated to ensure that they follow the
                   correct format
                if DATE_OF_BIRTH.validate() and EMAIL.validate() AND PHONE_NUMBER.validate()
                then    // FEES.validate() and DURATION.validate() are
                           user_defined functions
                        // They return a boolean value
                        // Depending on the parameter, they carry out the required validation
                           checks

                        // Generating a unique identifier for each student made. This will be
                           done in 3 steps
                        // asciiValue() returns the equivalent ASCII value for that character
                        // The value returned is of type integer
                        // STUDENT_NAME.firstCharacter() returns the first character of the
                           student STUDENT_NAME
                        PART_A = asciiValue(STUDENT_NAME.firstCharacter())

                        // From the DATE_OF_BIRTH entered in DD/MM/YYYY format, PART_B consists
                           of 'DD'
                        PART_B = BIRTHDAY_DATE

                        // FILE_CONTENTS.getLength() returns the number of student objects
                           present in the file
                        PART_C = FILE_CONTENTS.getLength() + 1

                        // Concatenating the 3 parts to obtain the final unique ID
                        // The string() function returns the value passed into it as a string
                        STUDENT_ID = string(PART_A) + string(PART_B) + string(PART_C)

                        // Creating a new student based on user input
                        // A student is created using OOP concepts, where each of the fields
                           entered are attributes of the student object, including the
                           student_id
                        CREATED_STUDENT = MAKE_STUDENT()

                        // writeFile("student_file.dat", CREATED_STUDENT) writes
                           CREATED_STUDENT to the file named "student_file.dat"
                        writeFile("student_file.dat", CREATED_STUDENT)

                else
                        // Date of birth, Email or Phone number validation failed. User must
                           reenter the values present in the field(s).
                        output "Error. Enter correct values for Date of birth, Email and
                               Phone number."
                end if
        end if
end procedure

*all .validate() procedures return True if the validation is successful and no error is found
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------

REGISTRATION

// Initialising class to make registration and term objects
MAKE_REGISTRATION = newClass()
MAKE_TERM = newClass()
```

```
// Function that registers a course, schedule and student based on user-input
procedure register(COURSE, SCHEDULE, STUDENT, START_DATE)
        // User chooses from previously created courses, schedules and students to register
           them
        // Checking if either of the required fields are blank
        // The user can only register a student if all fields are filled
        if COURSE = "" or
           SCHEDULE = "" or
           STUDENT = "" or
           START_DATE = "" then
                output "Error. Fill all required fields."       // Some fields have been left
                                                                   blank
        else
                // All fields have been filled
                // Start Date needs to be validated to ensure that it follow the correct format
                if START_DATE.validate() then   // START_DATE.validate() is a user_defined
                                                   function that returns a boolean value
                                                // It checks if the date entered is of a valid
                                                   format, and whether it matches the day of the
                                                   week for the schedule object entered

                        // Calling the calculate_dates() procedure to determine the future dates
                           of operations
                        FUTURE_DATES_ARRAY = calculate_dates()

                        // Generating a unique identifier for each registration
                        // Concatenating the 3 IDs from the Course, Schedule and Student entered
                           to obtain the final unique ID
                        // The 3 IDs are already in string form
                        REGISTRATION_ID = COURSE_ID + SCHEDULE_ID + STUDENT_ID

                        // Creating a new registration based on user input
                        // A registration is created using OOP concepts, where each of the fields
                           entered are attributes of the registration object, including the

                           registration_id and future dates
                        CREATED_REGISTRATION = MAKE_REGISTRATION()

                        // writeFile("register_file.dat", CREATED_REGISTRATION) writes
                           CREATED_REGISTRATION to the file named "register_file.dat"
                        writeFile("register_file.dat", CREATED_REGISTRATION)

                        // User is given an option to enter term details for the latest created
                           term. This part of the application is completely optional
                        input END_DATE
                        input DISCOUNT_TYPE
                        input DISCOUNT_AMOUNT
                        input PAYMENT_METHOD
                        input AMOUNT_PAID
                        input PAYMENT_DATE
                        input ACCOUNT_NAME
                        input STATUS

                        // Checking if either of the required fields are blank
                        // The user can only register a student if all fields are filled
                        if END_DATE = "" then
                                output "Error. Fill all required fields."       // Some fields
                                                                                   have been
                                                                                   left blank
                        else
                                // The validation is carried out only for those fields that have
                                   been entered
                                if END_DATE.validate() or DISCOUNT.validate() or
                                   AMOUNT_PAID.validate()
                                   or DISCOUNT_AMOUNT.validate()or PAYMENT_DATE.validate() then
                                        // Every time a new registration occurs, the term id for
                                           the first term created is set to '01'
                                        TERM_ID = 1

                                        // Creating a new term based on user input
                                        // A term is created using OOP concepts, where each of the
                                           fields entered are attributes of the term object,
                                           including specific details for the course, schedule and
                                           students created along with the term_id and
                                           registration_id
                                        CREATED_TERM = MAKE_TERM()

                                        // writeFile("term_file.dat", CREATED_TERM) writes
                                           CREATED_TERM to the file named "term_file.dat"
                                        writeFile("term_file.dat", CREATED_TERM)
                                else
                                        // The validations have failed. User must reenter the values
                                           present in the field(s).
                                        output "Error. Enter correct values."
                                end if
                        end if
                else
                        // Start date validation failed. User must reenter the values present in the
                           field(s).
                        output "Error. Enter correct values for Start Date."
                end if
        end if
end procedure
```

```
// Function to calculate the future days that the registered course will take place on
procedure calculate_dates()

        // Initialising an array that would store all future dates
        FUTURE_DATES_ARRAY = newArray()
        // DURATION refers to the field that was entered when the Course was created
        // The following loop runs for DURATION
        loop WEEK from 0 to DURATION - 1
                // The following operation is carried out in date format, not integer format
                NEXT_DAY = START_DATE + (7*WEEK)
                FUTURE_DATES_ARRAY.append(NEXT_DAY)
        end loop
        return FUTURE_DATES_ARRAY
end procedure

*all .validate() procedures return True if the validation is successful and no error is found
---------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
```

```
TERM

// Initialising class to make term object
MAKE_TERM = newClass()
```

```
// Function that displays terms based on 3 search criteria
procedure determine_term(SEARCH_STUDENT_NAME, SEARCH_COURSE_NAME, SEARCH_STATUS)
// User enters the above search criteria, based on which the term is identified.
   These are optional

        FILE_CONTENTS = readFile("term_file.dat")       // readFile("term_file.dat") reads
                                                           the contents of the term file and
                                                           returns them

        loop for each TERM in FILE_CONTENTS     // Looping through each term in FILE_CONTENTS
                // COURSE_NAME, STUDENT_NAME and STATUS are obtained from TERM using
                   OOP concepts
                if SEARCH_COURSE_NAME = COURSE_NAME then
                        if SEARCH_STUDENT_NAME = STUDENT_NAME then
                                if SEARCH_STATUS = STATUS then
                                        output TERM     // Diplaying terms that match the
                                                           criteria
                                end if
                        end if
                end if
        end loop
end procedure

// Function that allows users to edit term details for a specific term
procedure edit_term(TERM, DURATION, END_DATE, DISCOUNT_TYPE, DISCOUNT_AMOUNT, PAYMENT_METHOD,
                    AMOUNT_PAID, PAYMENT_DATE, ACCOUNT_NAME, STATUS)
        // User selects a term that they want to edit
        // User is given an option to enter term details for the latest created term. This
           part of the application is completely optional
        // The validation is carried out only for those fields that have been entered
        if DURATION.validate() or END_DATE.validate() or DISCOUNT.validate() or
           AMOUNT_PAID.validate()
           or DISCOUNT_AMOUNT.validate()or PAYMENT_DATE.validate() then
                FUTURE_DATES = calculate_dates()        // Calling the calculate_dates()
                                                           procedure to determine the
                                                           future dates of operations
                EDIT_TERM = TERM.editTerm()     // TERM.editTerm() returns term after
                                                   overwriting previous details
                                                   of TERM object with new user-input
                                                   details using OOP principles

                // writeFile("term_file.dat", EDIT_TERM) writes EDIT_TERM to the file named
                   "term_file.dat"
                writeFile("term_file.dat", EDIT_TERM)
        else
                // Validations failed. User must reenter the values present in the field(s).
                output "Error. Enter correct values."
        end if
end procedure

// Function to calculate the future days that the registered course will take place on
procedure calculate_dates()
        // Initialising an array that would store all future dates
        FUTURE_DATES_ARRAY = newArray()
        // DURATION refers to the new user-input that signifies the number of sessions
        // The following loop runs for DURATION
        loop WEEK from 0 to DURATION - 1
                // The following operation is carried out in date format, not integer format
                NEXT_DAY = START_DATE + (7*WEEK)
                FUTURE_DATES_ARRAY.append(NEXT_DAY)
        end loop
        return FUTURE_DATES_ARRAY
end procedure

// Function to add a new term if the status of a term is changed from pending to paid
procedure autogenerate_term()
        // TERM_STATUS refers to the status field for the term 'TERM'
        // TERM_STATUS.isChanged("Pending", "Paid") returns True if the value of TERM_STATUS
           is changed from "Pending" to "Paid"
        if TERM_STATUS.isChanged("Pending", "Paid") then
                // For the new term that will be created, it will start  7 days after the
                   current term ends
                NEW_TERM_START_DATE = TERM_END_DATE + 7
                // Accordingly, the future dates for the new term will be calculated
                NEW_TERM_FUTURE_DATES = calculate_dates()       // Calling the calculate_dates()
                                                                   procedure to determine the
                                                                   future dates of operations

                // A term is created using OOP concepts, where each of the fields entered are
                   attributes of the term object
                NEW_TERM = MAKE_TERM()

                // writeFile("term_file.dat", NEW_TERM) writes NEW_TERM to the file named
                   "term_file.dat"
                writeFile("term_file.dat", NEW_TERM)
        end if
end procedure

*all .validate() procedures return True if the validation is successful and no error is found
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------

REPORT

// Function to show details of courses, schedules and students
procedure display_details(SEARCH_CRITERIA)
        // User provides a search criteria based on which details are displayed

        if SEARCH_CRITERIA = "COURSE" then
                // readFile("course_file.dat") reads the contents of the course file and
                   returns them
                COURSE_FILE_CONTENTS = readFile("course_file.dat")
                loop for each COURSE in COURSE_FILE_CONTENTS    // Looping through each course
                                                                   present in the course file
                        output COURSE_DETAILS   // Displaying the details of COURSE
                end loop
        else if SEARCH_CRITERIA = "SCHEDULE" then
                // readFile("schedule_file.dat") reads the contents of the schedule file and
                   returns them
                SCHEDULE_FILE_CONTENTS = readFile("schedule_file.dat")
                loop for each SCHEDULE in SCHEDULE_FILE_CONTENTS        // Looping through each
                                                                           schedule present in
                                                                           the schedule file
                        output SCHEDULE_DETAILS // Displaying the details of SCHEDULE
                end loop
        else
                // readFile("student_file.dat") reads the contents of the student file and
                   returns them
                STUDENT_FILE_CONTENTS = readFile("student_file.dat")
                loop for each STUDENT in STUDENT_FILE_CONTENTS  // Looping through each student
                                                                  present in the student file
                        output STUDENT_DETAILS  // Displaying the details of STUDENT
                end loop
        end if
end procedure

// Function to save details in a text file so that users can read them and conver to other
   useful file formats
procedure generate_report(GENERATOR_CRITERIA)
        // User provides a field for which they want to save details in a text file

        if GENERATOR_CRITERIA = "TERM" then
                // readFile("term_file.dat") reads the contents of the term file and returns them
                TERM_FILE_CONTENTS = readFile("term_file.dat")
                // writeFile("term_report_file.txt", TERM_FILE_CONTENTS) writes TERM_FILE_CONTENTS
                  to the file named "term_report_file.txt"
                writeFile("term_report_file.txt", TERM_FILE_CONTENTS)
        else if GENERATOR_CRITERIA = "COURSE" then
                // readFile("course_file.dat") reads the contents of the course file and
                   returns them
                COURSE_FILE_CONTENTS = readFile("course_file.dat")
                // writeFile("course_report_file.txt", COURSE_FILE_CONTENTS) writes
                   COURSE_FILE_CONTENTS to the file named "course_report_file.txt"
                writeFile("course_report_file.txt", COURSE_FILE_CONTENTS)
        else if GENERATOR_CRITERIA = "SCHEDULE" then
                // readFile("schedule_file.dat") reads the contents of the schedule file and
                   returns them
                SCHEDULE_FILE_CONTENTS = readFile("schedule_file.dat")
                // writeFile("schedule_report_file.txt", SCHEDULE_FILE_CONTENTS) writes
                   SCHEDULE_FILE_CONTENTS to the file named "schedule_report_file.txt"
                writeFile("schedule_report_file.txt", SCHEDULE_FILE_CONTENTS)
        else

                // readFile("student_file.dat") reads the contents of the student file and
                   returns them
                STUDENT_FILE_CONTENTS = readFile("student_file.dat")
                // writeFile("student_report_file.txt", STUDENT_FILE_CONTENTS) writes
                   STUDENT_FILE_CONTENTS to the file named "student_report_file.txt"
                writeFile("student_report_file.txt", STUDENT_FILE_CONTENTS)
        end if
end procedure
```