Criterion C

1. Users can login as either a counsellor or a student

Once the program is run, we load the data saved from previous uses from our files.

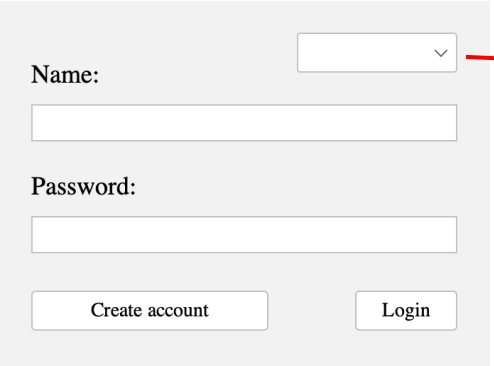
```java
public Login() {
    initComponents();
    Student.loadFromFile();
    Meeting.loadFromFile();
    jTextField1.setText("");
    jTextField2.setText("");
}
```

```java
public static synchronized void loadFromFile() {
    File db = new File("StuList8.txt");
    if (!db.exists() || !db.isFile()) {
        return;
    }

    FileInputStream f = null;
    ObjectInputStream o = null;
    try {
        f = new FileInputStream(db);
        o = new ObjectInputStream(f);
        Student.stuList = (LinkedList<Student>) o.readObject();
    } catch (IOException e) {
        System.out.println("Could not read student list from file.");
    } catch (ClassNotFoundException e) {
        throw new RuntimeException(e);
    } finally {
        try {
            if (o != null) {
                o.close();
            }
            if (f != null) {
                f.close();
            }
        } catch (IOException e) {
            System.out.println("Could not close input streams.");
        }
    }
}
```

The function loadFromFile() reads the objects from the file StuList8.txt and adds them to the LinkedList stuList.

The same is done for the function loadFromFile() for the Meetings, but we read from the file AllMeetList8.txt

We check first whether the user has selected a counsellor or student account:




```java
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    status = (String)jComboBox1.getSelectedItem();
```

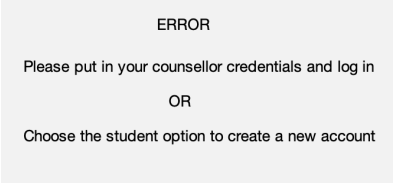
If they press "Create account", the following happens:

```java
// COUNSELLOR ERROR
if (status.equals("Counsellor")) {
    errorLogIn.setVisible(true);

// STUDENT NEW ACCOUNT
} else if (status.equals("Student")) {
    stuCurrent = new Student(jTextField1.getText(), jTextField2.getText());
    Student.saveToFile();
    Login.stuCurrent.meetList.remove(Login.stuCurrent.cancelledMeet);
    mainPageS.setVisible(true);
    this.setVisible(false);
}
```

New data is saved

b

The MainPage of the Student is then activated.



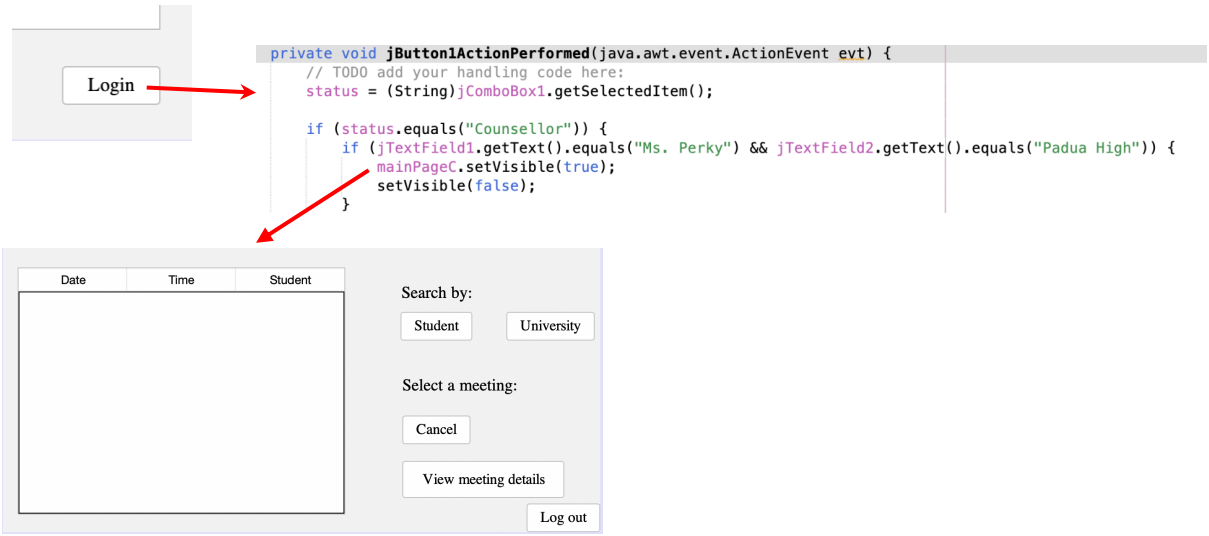
a

If they are a counsellor, this will lead to an error window popping up, since this program is meant to be used by only one counsellor with a set username and password. (a) But if they are a student, a new Student object will be created and added to a Linked List of all students.

This list will then be written to a file so that these users' data is not lost when the program is closed. (b)

Linked Lists are an appropriate data structure since they enable us to create a list without knowing the exact size of it, which is necessary for this program since we do not know how many students are going to be using it. Additionally, it provides an easy way to iterate through the Students stored inside the list when we want to use it for other functions later on.

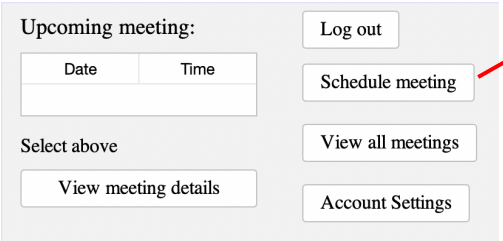
However, if they decide to log in directly, the counsellor's credentials are checked against the set values:



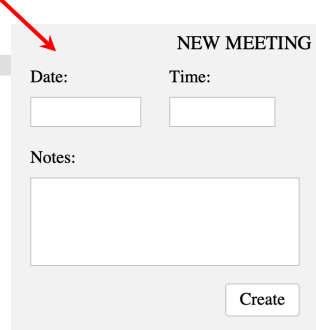
While for the students the inputted values in the TextFields are checked against all the existing Student objects.

```
} else if (status.equals("Student")) {
    for (int i = 0; i < Student.stuList.size(); i++) {
        if (Student.stuList.get(i).getUsName().equals(jTextField1.getText()) &&
        Student.stuList.get(i).getPassWrd().equals(jTextField2.getText())) {
            stuCurrent = Student.stuList.get(i);
            mainPageS.setVisible(true);
            setVisible(false);

        }
    }
}
```

2. Students can schedule meetings and input details in a note section




```java
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // NEW MEETING
    newMeet.setVisible(true);
    jTextField1.setText("");
    jTextField2.setText("");
    jTextArea1.setText("");
}
```



Once the student has inputted all of the details for scheduling their meeting, they can press the "create" button.


```java
private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {
    // NEW MEETING
    Meeting temp = new Meeting(Login.stuCurrent, jTextField1.getText(),
    jTextField2.getText(), jTextArea1.getText());
    Login.stuCurrent.addMeet(temp);

    newMeet.setVisible(false);
    renderMeetingList();
    Student.saveToFile();
    Meeting.saveToFile();
}
```

A new Meeting is created using the user's inputs, which is then added to the student's list of meetings.

We then render the meeting list, so that the "Upcoming meeting" is updated. And we save the data to file

With the function .addMeet(), the meeting sorted by date:

```java
public void addMeet(Meeting meet) {
    meetList.add(meet);
    Collections.sort(meetList);
}
```

When an instance of the Student class is made, they get assigned a Linked List where all of the meetings that student schedules are stored.

```java
public class Student implements Serializable {

    private String usName;
    private String passWrd;
    private LinkedList<Meeting> meetList;
    public  LinkedList<String> uniList;
    public static LinkedList<Student> stuList = new LinkedList<Student>();
    public Meeting cancelledMeet;

    public Student(String usName, String passWrd){
        this.usName = usName;
        this.passWrd = passWrd;
        this.meetList = new LinkedList<Meeting>();
        this.uniList = new LinkedList<String>();
        stuList.add(this);
    }
```

In addition we do not need to also add the meeting manually to the list of all meetings created

by any student and sort it, since in the Meeting constructor it is done so automatically.

```
public Meeting(Student stu, String date, String time, String notes) {
    this.stu = stu;
    this.notes = notes;
    this.date = date;
    this.time = time;
    this.status = "pending";

    allMeet.add(this);
    Collections.sort(allMeet);
}
```

The two separate lists of allMeet and meetList are necessary so that we can easily display the corresponding meetings in the MainPageCounsellor and the MainPageStudent. In addition, sorting them as we add more elements prevents us from having to go through that process everytime we want to display the upcoming lists.

If they later want to view the meeting details or edit the meeting notes, they can select the upcoming meeting and click on "View meeting details"

The counsellor is also able to look at these meeting details through the MainPageCounsellor:

View meeting details

```java
private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Meeting currentD = (Meeting)jTable1.getValueAt(jTable1.getSelectedRow(), jTable1.getSelectedColumn());
    meetDet.setVisible(true);
    jTextField2.setText(currentD.getDate());
    jTextField3.setText(currentD.getTime());
    jTextField1.setText(currentD.getStu().getUsName());
    jTextArea2.setText(currentD.getNotes());
}
```

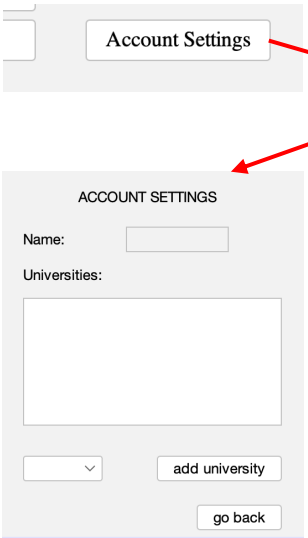
Date: Time: Student:

Student notes:

Go back

3. Students can select tags relating to which universities they want to go to

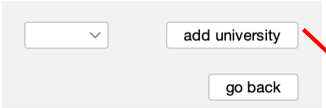
On MainPageStudent, students can access their account settings:




```java
private void jButton9ActionPerformed(java.awt.event.ActionEvent evt) {
    // ACCOUNT SETTINGS
    accSettings.setVisible(true);

    // display information
    jTextField3.setText(Login.stuCurrent.getUsName());
    jTextArea5.setText("");
    for (int i = 0; i < Login.stuCurrent.uniList.size(); i++) {
        jTextArea5.setText(jTextArea5.getText() + Login.stuCurrent.uniList.get(i) + "\n");
    }
}
```

When the button is pressed, a window with the account settings is displayed, including a combo box that lists the student's chosen universities.

If they want to add more universities, they can go to the ComboBox below the list and click on the one they want to add. The ComboBox contains the top 25 universities in the UK[1].



By clicking "add university", the selected item of the ComboBox will be added to the uniList of the student. The TextArea is also updated to display the new addition.

```java
private void jButton11ActionPerformed(java.awt.event.ActionEvent evt) {
    // ADD UNIVERSITY
    boolean flag = true;

    // check if the university has already been added
    String selected = (String)jComboBox1.getSelectedItem();
    for (int i = 0; i < Login.stuCurrent.uniList.size(); i++) {
        if (selected.equals(Login.stuCurrent.uniList.get(i))) {
            flag = false;
        }
    }

    // add university and display
    if (flag) {
        Login.stuCurrent.uniList.add((String)jComboBox1.getSelectedItem());
        jTextArea5.setText("");
        for (int i = 0; i < Login.stuCurrent.uniList.size(); i++) {
            jTextArea5.setText(jTextArea5.getText() + Login.stuCurrent.uniList.get(i) + "\n");
        }
    }

}
```

[1] "Best Global Universities in United Kingdom," U.S. News, https://www.usnews.com/education/best-global-universities/united-kingdom.

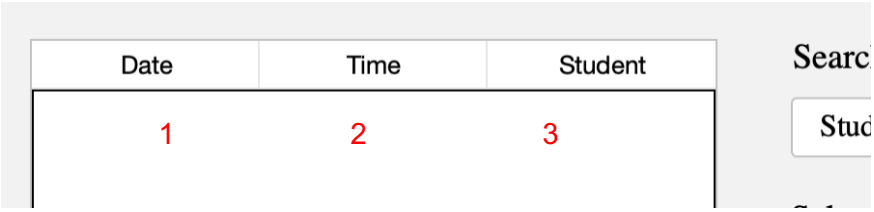
```java
public class Student implements Serializable {

    private String usName;
    private String passWrd;
    private LinkedList<Meeting> meetList;
    public  LinkedList<String> uniList;
    public static LinkedList<Student> stuList = new LinkedList<Student>();
    public Meeting cancelledMeet;

    public Student(String usName, String passWrd){
        this.usName = usName;
        this.passWrd = passWrd;
        this.meetList = new LinkedList<Meeting>();
        this.uniList = new LinkedList<String>();
        stuList.add(this);
    }
```

The student's uniList is unique to each instantiation of the Student class. Similarly to the meetList, a new list of the student's universities is created with each student.

4. Counsellors and students can view all meetings in an "upcoming" list

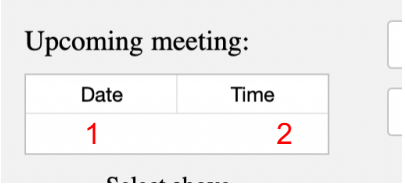
For counsellors, all the upcoming meetings are displayed in a table organised chronologically. Since allMeet is already sorted, when the MainPageCounsellor is activated, we loop through the list and display the information in the table using the function renderMeetingList().

```
public MainPageCounsellor(Login login) {
    this.login = login;
    initComponents();

    // design aspects
    jTable1.setShowGrid(true);
    jTable1.setGridColor(Color.black);

    // upcoming list
    renderMeetingList();
    Meeting.saveToFile();
}

private void renderMeetingList() {
    for (int i = 0; i < Meeting.allMeet.size(); i++) {
        jTable1.setValueAt(Meeting.allMeet.get(i).getDate(), i, 0);        1
        jTable1.setValueAt(Meeting.allMeet.get(i).getTime(), i, 1);        2
        jTable1.setValueAt(Meeting.allMeet.get(i).getStu().getUsName(), i, 2);   3
    }
}
```



For students, we only display the first upcoming meeting on the Main Page, since they are likely to only have one meeting scheduled with the counsellor. We display it the same way as for the MainPageCounsellor above, but without the student name.

```
public MainPageStudent(Login login) {
    this.login = login;
    initComponents();

    // design aspects
    jTable1.setShowGrid(true);
    jTable1.setGridColor(Color.black);

    // upcoming list
    clearMeetingList();
    renderMeetingList();
    cancelMessage();
}
```




```
private void renderMeetingList() {
    if (Login.stuCurrent != null && !Login.stuCurrent.meetList.isEmpty()) {
        jTable1.setValueAt(Login.stuCurrent.meetList.get(0).getDate(), 0, 0);   1
        jTable1.setValueAt(Login.stuCurrent.meetList.get(0).getTime(), 0, 1);   2
    }
}
```

Additionally, if the student wants to see all of their planned meetings, they can press on the view all meetings button. It will open a new window with the same table format as previously, but displaying all the meetings this time.

Here we use the function renderMeetingList2() which loops through the student's meeting list and displays the information in the table.

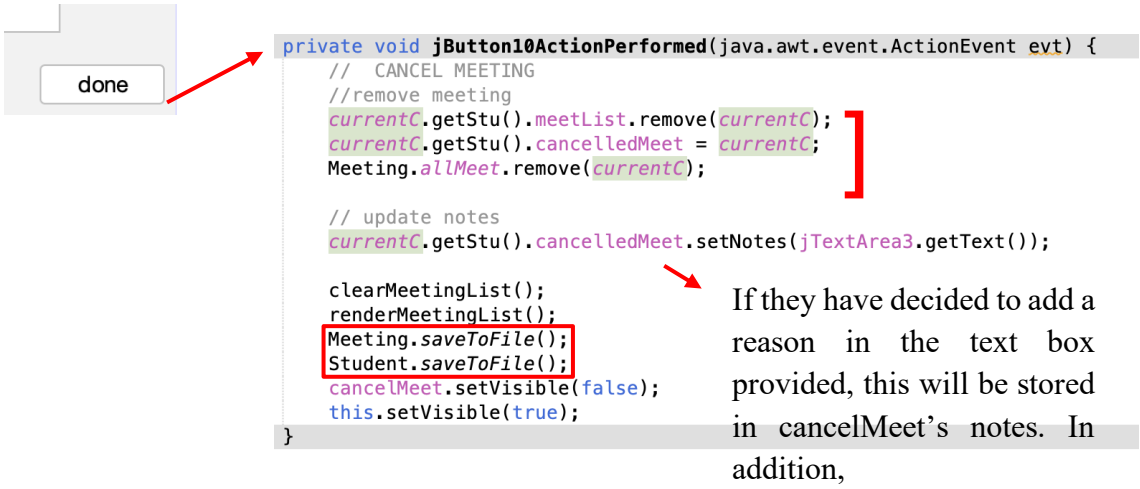
```
private void renderMeetingList2() {
    if (Login.stuCurrent != null && !Login.stuCurrent.meetList.isEmpty()) {
        for (int i = 0; i < Login.stuCurrent.meetList.size(); i++) {
            jTable3.setValueAt(Login.stuCurrent.meetList.get(i).getDate(), i, 0);
            jTable3.setValueAt(Login.stuCurrent.meetList.get(i).getTime(), i, 1);
            jTable3.setValueAt(Login.stuCurrent.meetList.get(i).getNotes(), i, 2);
        }
    }
}
```

5. **Counsellor can cancel meetings, with the student receiving a warning about the update**

If the counsellor wants to cancel a meeting, they have to select one of the rows of the desired meeting and press the "cancel" button. Once they do this, a new window will appear called cancelMeet, displaying all the necessary information about the meeting.


```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // CANCEL MEETING
    cancelMeet.setVisible(true);

    // display information
    currentC = Meeting.allMeet.get(jTable1.getSelectedRow());
    jTextField4.setText(currentC.getDate());
    jTextField5.setText(currentC.getTime());
    jTextField6.setText(currentC.getStu().getUsName());
```

Once the counsellor presses the "done" button, this action will remove the meeting from the total list of meetings and the student's meeting list, and store the meeting in a variable called cancelMeet in the Student class.




```
private void jButton10ActionPerformed(java.awt.event.ActionEvent evt) {
    //  CANCEL MEETING
    //remove meeting
    currentC.getStu().meetList.remove(currentC);
    currentC.getStu().cancelledMeet = currentC;
    Meeting.allMeet.remove(currentC);

    // update notes
    currentC.getStu().cancelledMeet.setNotes(jTextArea3.getText());

    clearMeetingList();
    renderMeetingList();
    Meeting.saveToFile();
    Student.saveToFile();
    cancelMeet.setVisible(false);
    this.setVisible(true);
}
```

We also need to now update the Files containing the student and the meeting lists. We can do this using the saveToFile() functions. Which writes all the objects in the list into the corresponding file.

Meeting class:

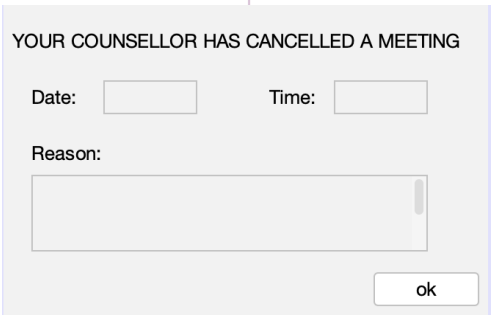
```java
public static synchronized void saveToFile() {
    File db = new File("AllMeetList8.txt");
    if (db.exists()) {
        db.delete();
    }

    FileOutputStream f = null;
    ObjectOutputStream o = null;

    try {
        // CREATE FILE STREAM
        f = new FileOutputStream(db);
        o = new ObjectOutputStream(f);
        o.writeObject(allMeet);

        // CLOSE FILE STREAM
        o.close();
        f.close();
        System.out.println("Saved to file successfully.");
    } catch (IOException e) {
        System.out.println("Could not write to file.");
    } finally {
        try {
            if (o != null) {
                o.close();
            }
            if (f != null) {
                f.close();
            }
        } catch (IOException e) {
            System.out.println("Could not close output streams.");
        }
        System.out.println("Finished saving to file.");
    }
}
```

Student class:

```java
public static synchronized void saveToFile() {
    File db = new File("StuList8.txt");
    if (db.exists()) {
        db.delete();
    }

    FileOutputStream f = null;
    ObjectOutputStream o = null;

    try {
        // CREATE FILE STREAM
        f = new FileOutputStream(db);
        o = new ObjectOutputStream(f);
        o.writeObject(stuList);

        // CLOSE FILE STREAM
        o.close();
        f.close();
        System.out.println("Saved to file successfully.");
    } catch (IOException e) {
        System.out.println("Could not write to file.");
    } finally {
        try {
            if (o != null) {
                o.close();
            }
            if (f != null) {
                f.close();
            }
        } catch (IOException e) {
            System.out.println("Could not close output streams.");
        }
        System.out.println("Finished saving to file.");
    }
}
```

In order to inform the student of this cancellation, when the MainPageStudent is activated we use the function cancelMessage(). This checks if cancelMeet is null. If it is not, then we display a window with the information of this cancelled meeting.

```java
public MainPageStudent(Login login) {
    this.login = login;
    initComponents();

    // design aspects
    jTable1.setShowGrid(true);
    jTable1.setGridColor(Color.black);

    // upcoming list
    clearMeetingList();
    renderMeetingList();
    cancelMessage();
}

private void cancelMessage() {
    if (Login.stuCurrent != null && Login.stuCurrent.cancelledMeet != null) {
        jTextField6.setText(Login.stuCurrent.cancelledMeet.getDate());
        jTextField7.setText(Login.stuCurrent.cancelledMeet.getTime());
        jTextArea4.setText(Login.stuCurrent.cancelledMeet.getNotes());
        counsCancel.setVisible(true);
    }
}
```

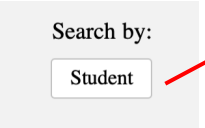


Once the student clicks on the “ok” to close this window, cancelMeet is set back to null.

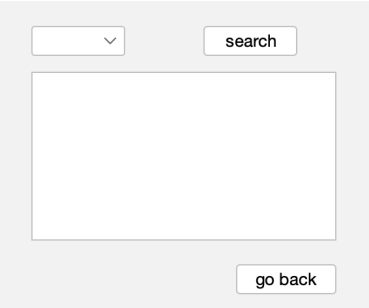
```java
private void jButton10ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    login.stuCurrent.cancelledMeet = null;
}
```

6. Counsellors can search scheduled meetings based on the student and universities

From the MainPageCounselor, the user can click on the “search by student” button that takes them to another window.

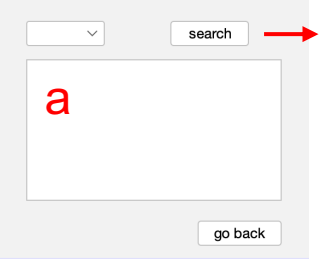



```java
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    // STUDENT SEARCH
    stuSearch.setVisible(true);

    // show all students in ComboBox
    jComboBox1.removeAllItems();
    for (int i = 0; i < Student.stuList.size(); i++) {
        jComboBox1.addItem(Student.stuList.get(i).getUsName());
    }
}
```



The student’s names are added to the ComboBox.

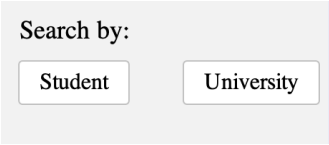
Once the counsellor has chosen the student they want to use to search from the ComboBox, they can click “search” and the meetings with that student will appear in the TextArea below.



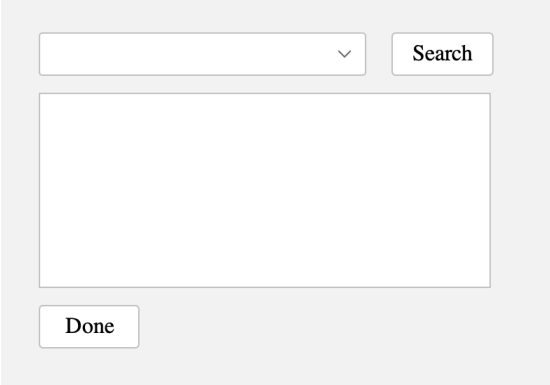

```java
private void jButton11ActionPerformed(java.awt.event.ActionEvent evt) {
    // STUDENT SEARCH
    jTextArea5.setText("");

    // search for selected student
    Student current = null;
    for (int i = 0; i < Student.stuList.size(); i++) {
        if (jComboBox1.getSelectedItem().equals(Student.stuList.get(i).getUsName())) {
            current = Student.stuList.get(i);
        }
    }

    // display their meetings
    for (int i = 0; i < current.meetList.size(); i++ ) {
        jTextArea5.setText(jTextArea5.getText() + current.meetList.get(i).getDate() + " - "
       + current.meetList.get(i).getTime() + "\n");
    }
}
```

a

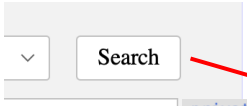
To search by university, they can go to the corresponding button, which displays another window.




```java
private void jButton9ActionPerformed(java.awt.event.ActionEvent evt) {
    // UNIVERSITY SEARCH
    uniSearch.setVisible(true);
    jTextArea6.setText("");
}
```



After choosing the university they want from the ComboBox, they click “Search”, and then each student’s university list is searched for the selected university in the ComboBox.




```java
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // UNIVERSITY SEARCH
    jTextArea6.setText("");
    String uniChoice = (String)jComboBox2.getSelectedItem();

    // display students w selected university
    for (int i = 0; i < Student.stuList.size(); i++) {
        for (int e = 0; e < Student.stuList.get(i).uniList.size(); e++) {
            if (Student.stuList.get(i).uniList.get(e).equals(uniChoice)) {
                jTextArea6.setText(jTextArea6.getText() + Student.stuList.get(i).getUsName()+ "\n");
            }
        }
    }
}
```

➔ Process of logging out

For logging out, both of the interfaces for the student and counsellor perform the same functions:

```java
private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
    // LOG OUT
    Student.saveToFile();
    Meeting.saveToFile();
    clearMeetingList();
    this.setVisible(false);
    login.setVisible(true);
}
```

All of the data is again saved to file, and the login page is set to visible again.

Word count: 855