GROUP ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

AAPP013-4-2-OOP-L-3

UCDF2304ICT(SE) / UCDF2304ICT / UCDF2304ICT(DI) / UCDF2304ICT(ITR)

Group: 7

Member 1: Elianna Catrina Herrera (TP073631) [Student Side]

Member 2: Ejjaz Hakimi bin Mohamad Azan (TP073318) [Lecturer Side]

HAND OUT DATE: 4 October 2024

HAND IN DATE: 18 December 2024

WEIGHTAGE: 100%

INSTRUCTIONS TO CANDIDATES:

1. Submit your assignment online in Moodle Folder unless advised otherwise
2. Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld
3. Cases of plagiarism will be penalized
4. You must obtain at least 50% in each component to pass this module

Table of Contents

1. Introduction

Project purpose

A Psychology consultation management system has been built for Asia Pacific University Innovation & Technology (APU) in order to streamline the process of booking and managing consultations between students and psychology lecturers that would enhance the effectiveness of management and increase user satisfaction. This can be further proven based on (Gaith Ibrahim Alshammare, 2022), where he mentioned that users will likely repeat the services provided when user expectations are met as quality service has its influential essence towards user satisfaction. Moreover, having an online consultation management could definitely waver users' decision to proceed with the consultation service as it is intuitive and easy to access.

Target Users

1) **APU Students**
   Students will be able to register, book consultation appointments and view their appointment details.
2) **Psychology Lecturers**
   Lecturers will have access to view and manage their appointment slots effectively.

Technology Used

The system will be implemented in Java with the aid of Apache NetBeans, a smart editor for Java, applying the object-oriented programming language and frameworks involved in building the system.

Functional Requirements

Student

1) Student can register an account at the registration page
2) Student can view a list of lecturer's availability and their consultation slots at the appointment page.
3) Student can cancel or reschedule a consultation slot at the appointment page.
4) Student can give feedback on their consultation experience after their session at the feedback page.
5) Student can book a consultation session with the lecturer at the Consultation page.

Lecturer

1) Lecturer will be able to register an account using the registration page.
2) Lecturer can manage their consultation slots by setting the available time and location.
3) Lecturer can view upcoming and past appointments to keep track of their schedules.
4) Lecturer can decide if rescheduling requests were to be approved or rejected by viewing the list of consultation sessions.
5) Lecturer can read the students feedback at the feedback page.

Non-functional Requirements

1) The system's interface must be intuitive and user-friendly for students and lecturers to utilize
2) Feedback forms must be simple for better user experience and ease of use.
3) The system must ensure that the details of appointments, cancellations and rescheduling actions are accurately recorded and updated into text files for further reference.

2. Data Text Files

Below are the text files that have been created to store and manage the student and lecturer's information.

consultation.txt

```
CN288:Tuesday:2024-12-10:12.30:13.30:E-05-01
CN118:Tuesday:2024-12-10:14.30:16.30:E-05-01
CN895:Friday:2024-12-13:10.30:11.30:E-05-01
CN325:Monday:2024-12-16:15.30:16.30:E-05-02
CN398:Tuesday:2024-12-17:13.30:14.30:E-05-03
CN219:Wednesday:2024-12-18:13.30:14.30:E-05-02
CN144:Thursday:2024-12-19:15.30:16.30:E-05-01
CN226:Friday:2024-12-20:15.30:16.30:E-05-03
CN617:Saturday:2024-12-21:12.30:13.30:E-05-01
CN978:Sunday:2024-12-22:12.30:13.30:E-05-02
CN657:Tuesday:2024-12-24:12.30:13.30:E-05-02
CN585:Wednesday:2024-12-25:12.30:13.30:E-05-02
CN300:Friday:2024-12-27:10.30:11.30:E-05-03
```

In the text file of consultation.txt, after lecturers create a consultation slot at the Consultation Screen, the data will be stored in the format of

consultID : consultDay : consultDate : consultStartTime : consultEndTime : consultLocation

booking.txt

```
SB610:Friday:2024-12-13:10.30:11.30:E-05-01:Booked
SB464:Tuesday:2024-12-17:13.30:14.30:E-05-03:Booked
SB868:Tuesday:2024-12-17:13.30:14.30:E-05-03:Booked
SB481:Tuesday:2024-12-17:13.30:14.30:E-05-03:Booked
SB692:Wednesday:2024-12-18:13.30:14.30:E-05-02:Booked
```

Based on the figure above, after students have booked their consultations at the booking page, the data will be stored in the format of:

bookID: bookDay : bookDate : bookStartTime : bookEndTime : bookLocation : bookStatus

feedback.txt

```
FB34:Friday:2024-12-21:14.30:15.30:E-05-02:Completed:good session:thank you!
```

In the figure above, after students submit their feedback at the feedback page, their data will be stored in the format of:

feedbackID : bookDay : bookDate : bookStartTime : bookEndTime : bookLocation : bookStatus : feedbackLecturer: feedbackStudent

lecturer.txt

```
LT586:Ejjaz:122:Male
LT867:Gaurav:344:Male
LT743:Ryan:567:Male
LT469:Tengku:456:Male
LT481:Elly:567:Female
LT231:Suchi:123:Female
LT757:Nashran:123:Male
```

After the lecturer registers themselves the lecturer's information will be stored in the lecturer.txt with the format of:

lecturerID : lecturerName : lecturerPassword : Gender

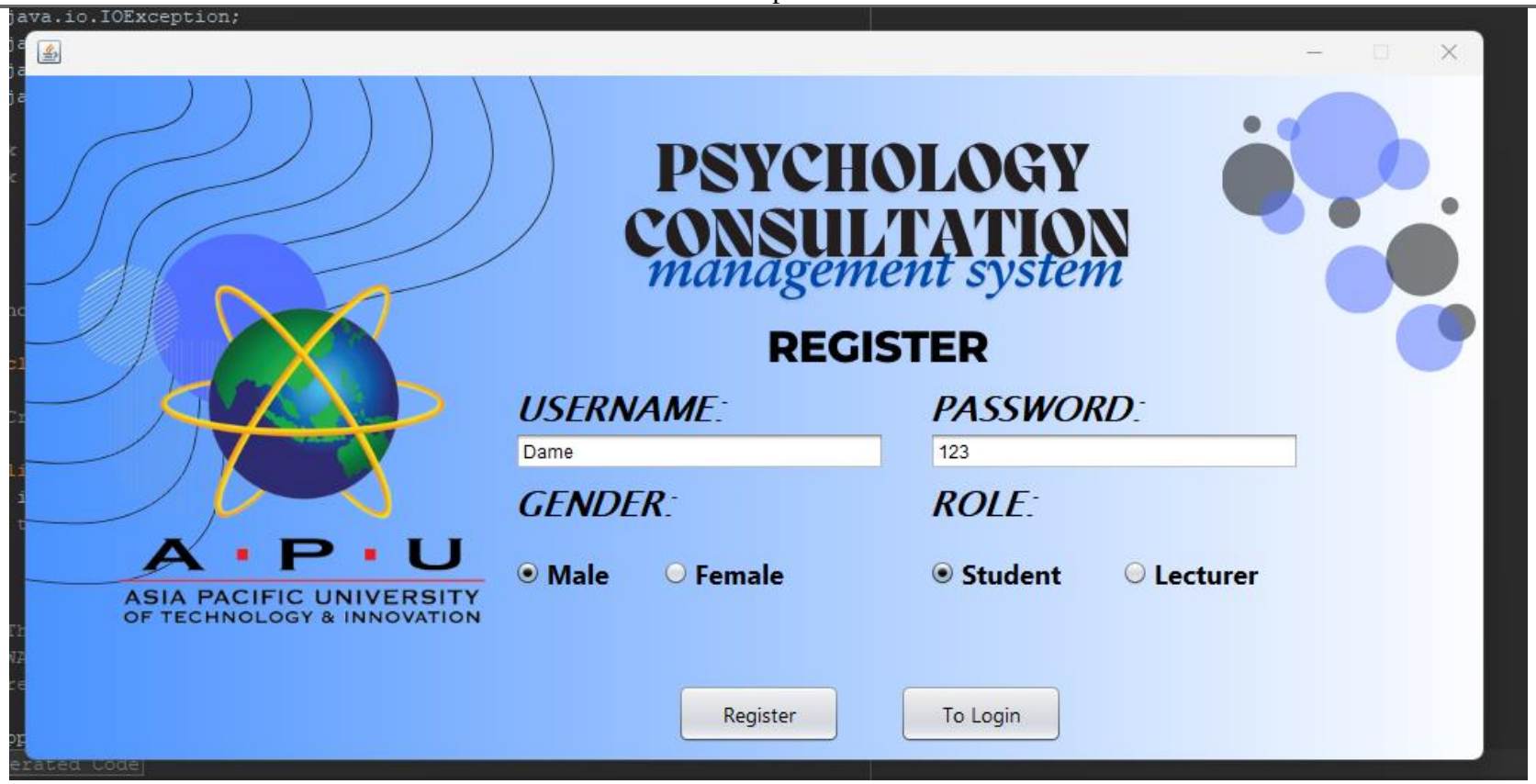
student.txt

```
ST898:Harris:123:Male
ST778:Winiee:123:Female
```

Students' details after registration will also be stored in the student.txt text file with the format of:

studentID : studentName : studentPassword : Gender

3. Sample Outputs of Program Execution

Scenario	Output
Student Registration	 User would need to input their desired username, password, gender as well as select 'student' as their role. Once this is done, user will have to click on the Register button.

If registration is successful, they will be notified of its success. If registration fails however (in the case that a record with a matching username exists) the system will display an error message notifying users of the registration failure which can be seen in the figure showcased below.

Once registration is successful, their registration record will be stored in student.txt seen below

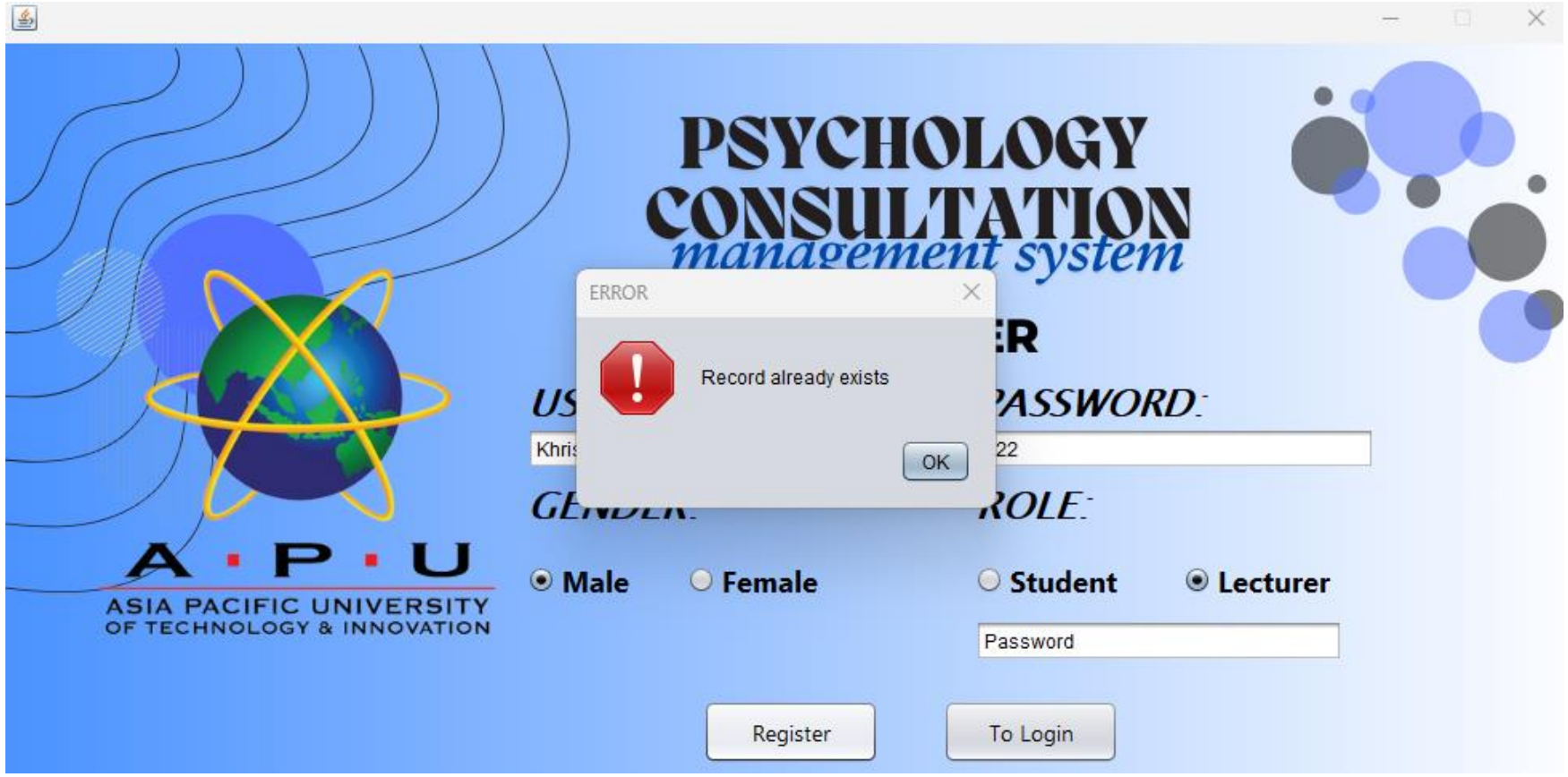
```
ST898:Harris:123:Male
ST778:Winiee:123:Female
ST439:Joe:124:Male
ST837:Dame:123:Male
```

Lecturer Registration	 If a user wishes to register as a lecturer however, they need to fill up the same fields as student registration with the exception that they select Lecturer as their role. For security measure, users will also have to input a master password before registration can be successful (the assumption is made that only lecturers will know the master password). If successful, the system will prompt the user notifying them of their success. If the master password is incorrect, an error prompt will be displayed notifying them of so. Both these instances can be seen in the figures below.


PSYCHOLOGY
CONSULTATION
management system
NOTICE
Register Succesful
OK
Male
Female
Student
Lecturer
Password
Register
To Login
A · P · U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION


PSYCHOLOGY
CONSULTATION
management system
ERROR
Master Password is Incorrect
OK
Male
Female
Student
Lecturer
skibidi
Register
To Login
A · P · U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

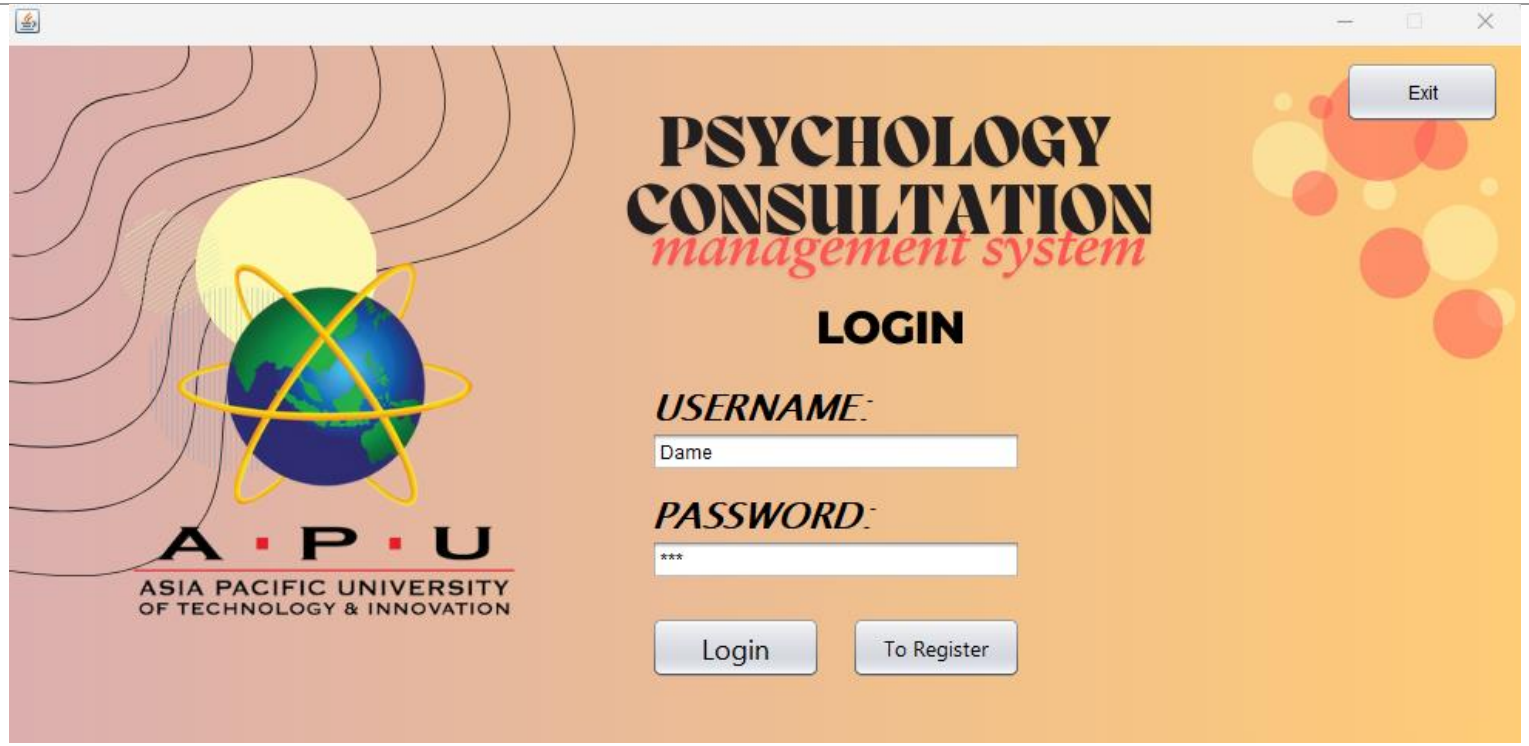
Like in student registration, the lecturer registration can fail if it detects that the record already exists



Once registration is successful, the account details will be stored in lecturer.txt seen below

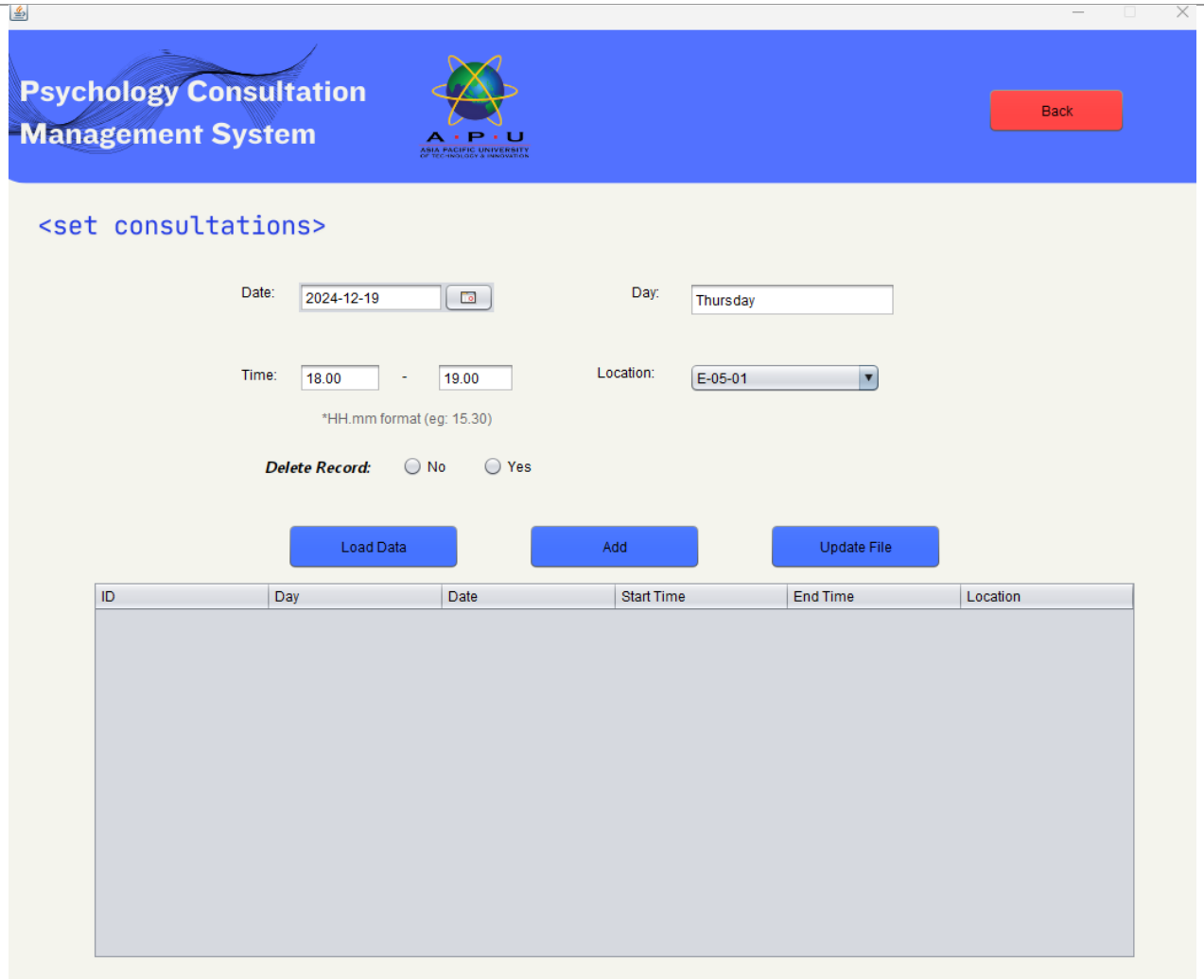
```
LT586:Ejjaz:122:Male
LT867:Gaurav:344:Male
LT743:Ryan:567:Male
LT469:Tengku:456:Male
LT481:Elly:567:Female
LT231:Suchi:123:Female
LT757:Nashran:123:Male
LT94:Lamar:124:Male
LT190:Khris:122:Male
```

Login

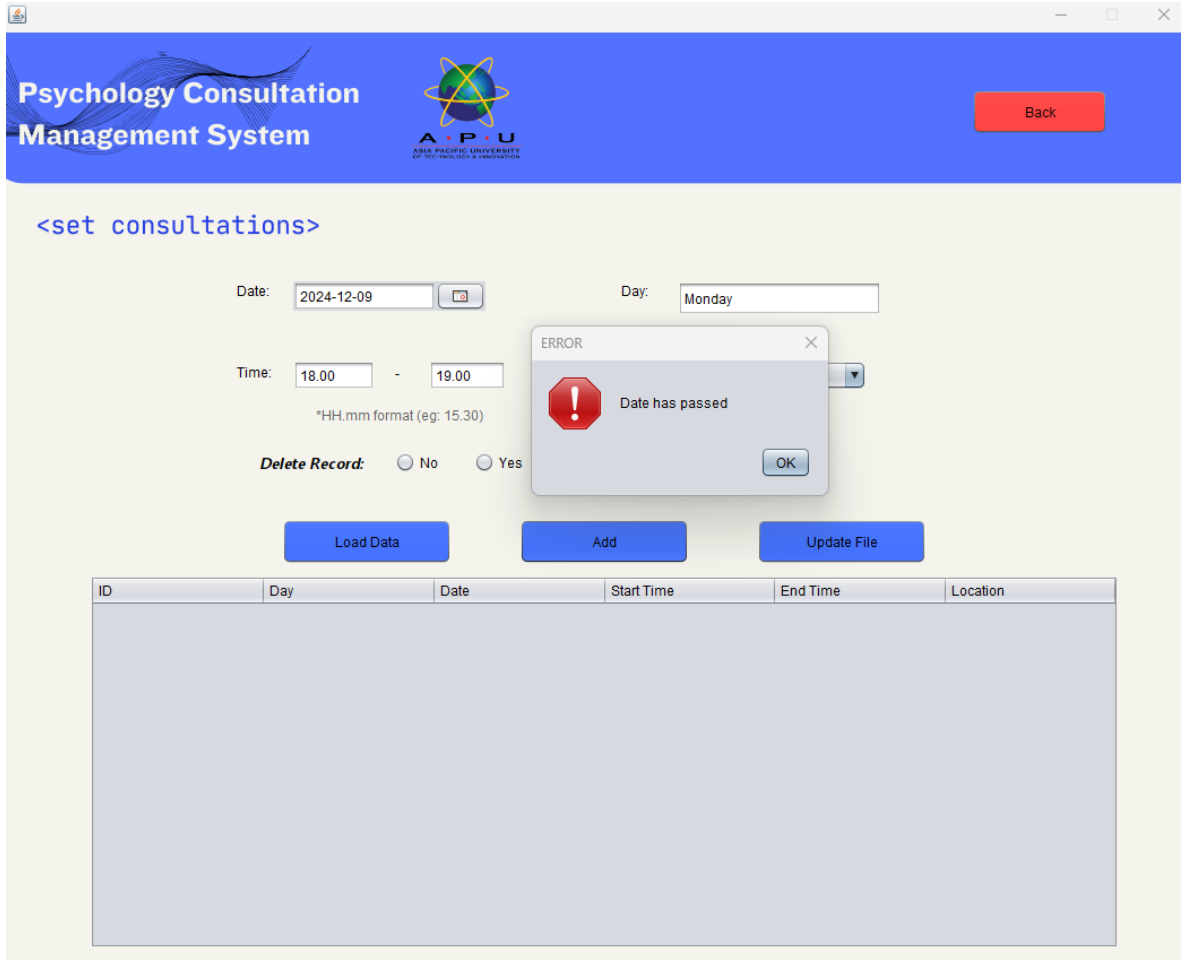


To login, users will have to input their username alongside their password and click on the login button. If the records are incorrected, they will be alerted of so and prompted to try again. If successful, they will then be alerted of its success and redirected to either the student homepage or lecturer homepage based on their role. These outputs can be seen in the figure below

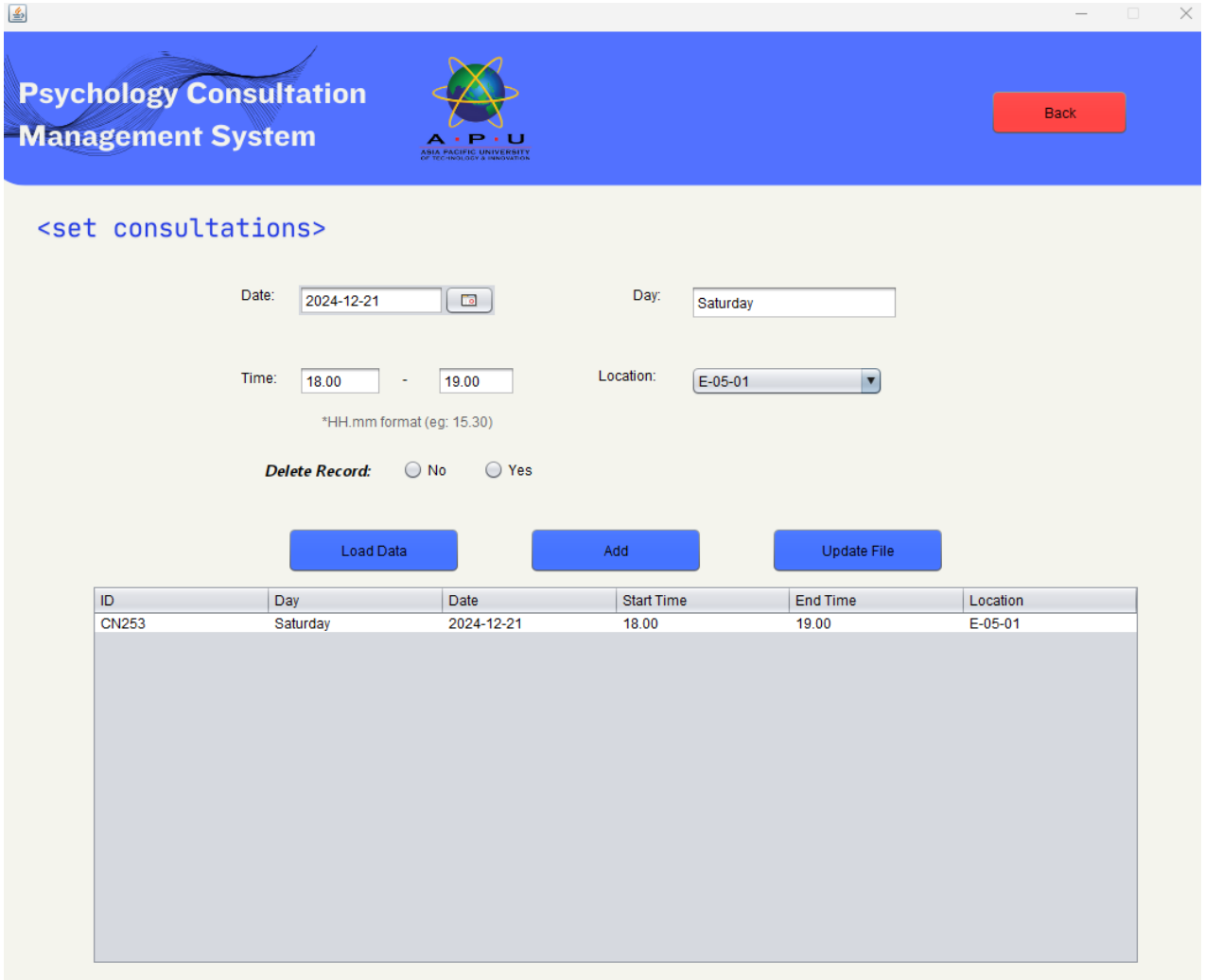
	
Set Consultation Slot	If a lecturer wishes to set a consultation slot, they must fill up the following details, the date of consultation, day (autofilled by the system based on the date selected using JCalender), starting & ending time of slot, as well as the location (a dropdown menu consisting of three values: E-05-01, E-05-02, E-05-03).

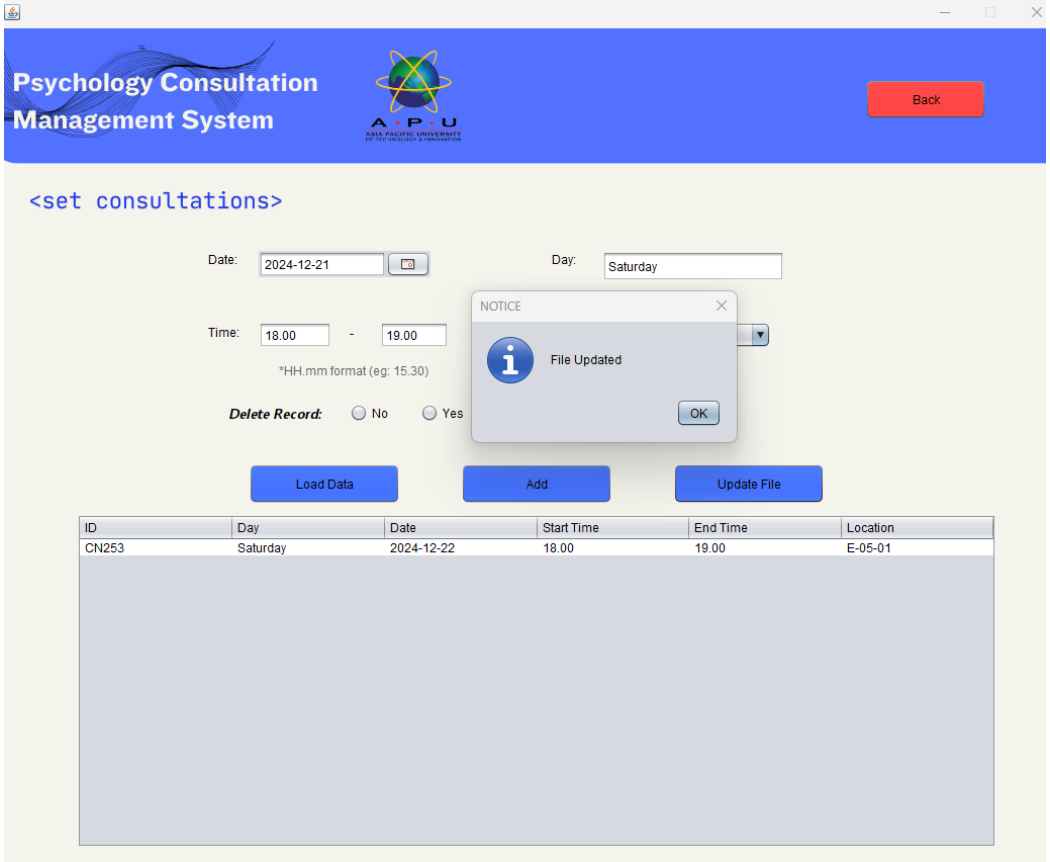

Psychology Consultation
Management System
A·P·U
Back
<set consultations>
Date:
2024-12-19
Day:
Thursday
Time:
18.00
-
19.00
Location:
E-05-01
*HH.mm format (eg: 15.30)
Delete Record:
No
Yes
Load Data
Add
Update File
ID
Day
Date
Start Time
End Time
Location

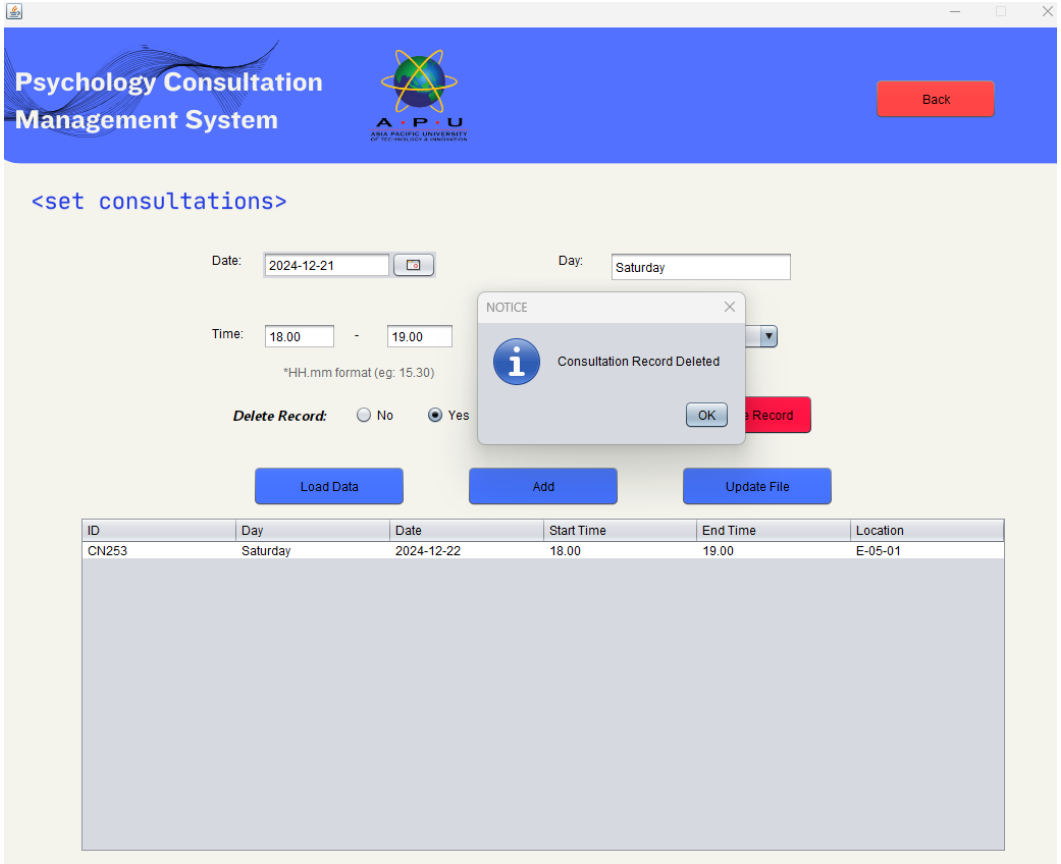
Validation of all the values exists that is the system will check if the date has already passed, whether the time has already passed (if date chosen is current day) and whether time range is valid (starting time is before ending time / starting time and ending time are different values). If any of these validation fails, the system will notify the lecturer and will not proceed with setting the consultation slot. An example of which can be seen below in which the date selected has already passed

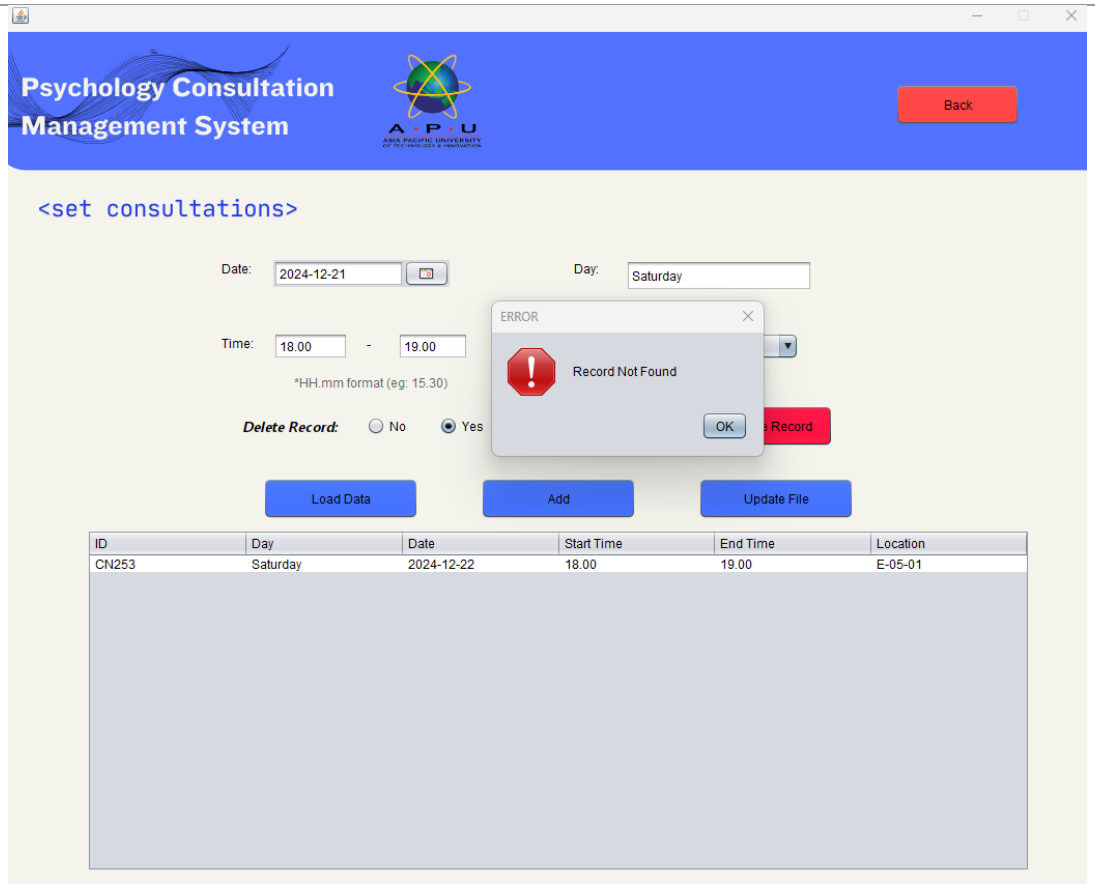


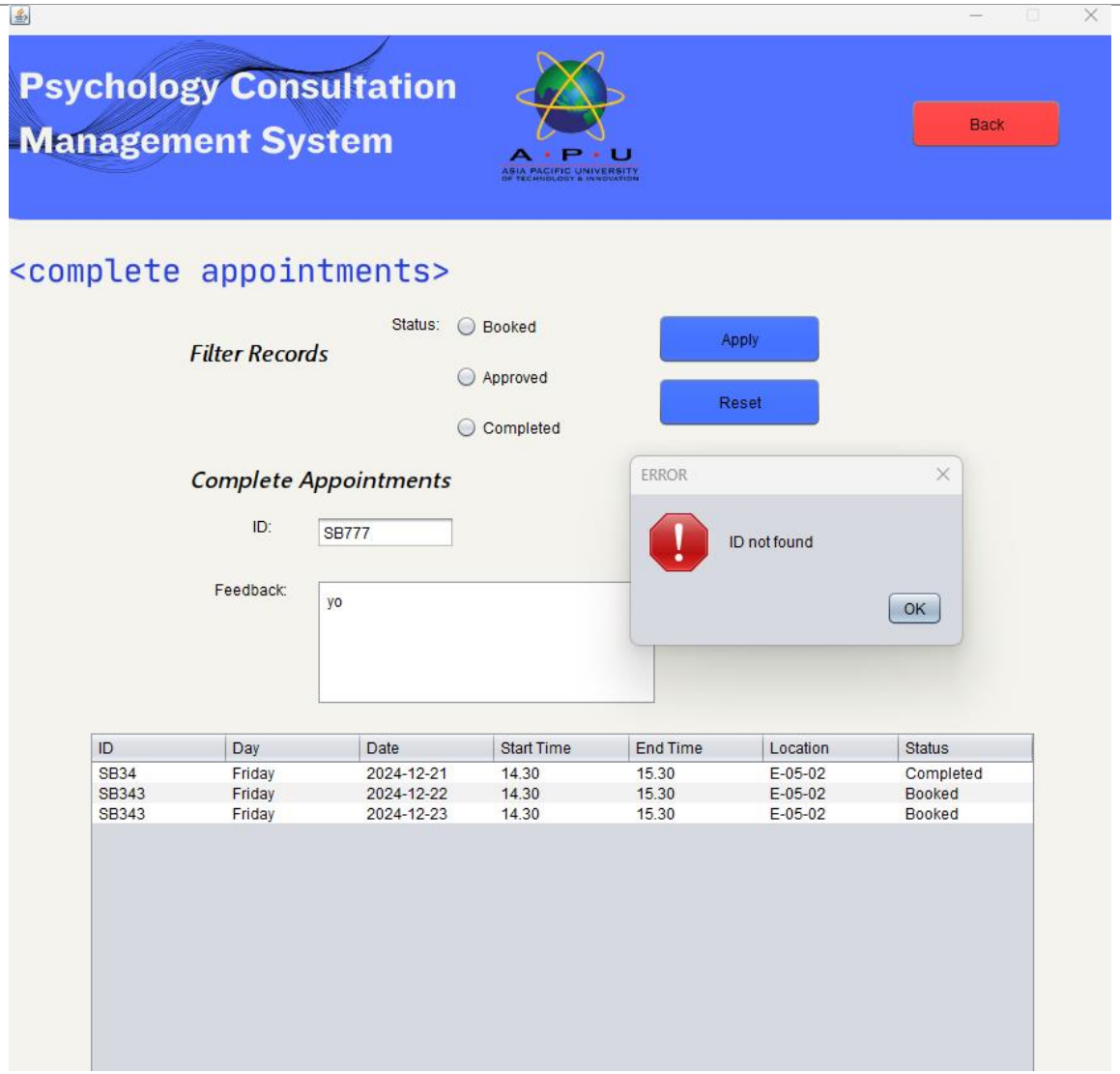
If validation is successful, the slot will be added into the table as well as in consultation.txt

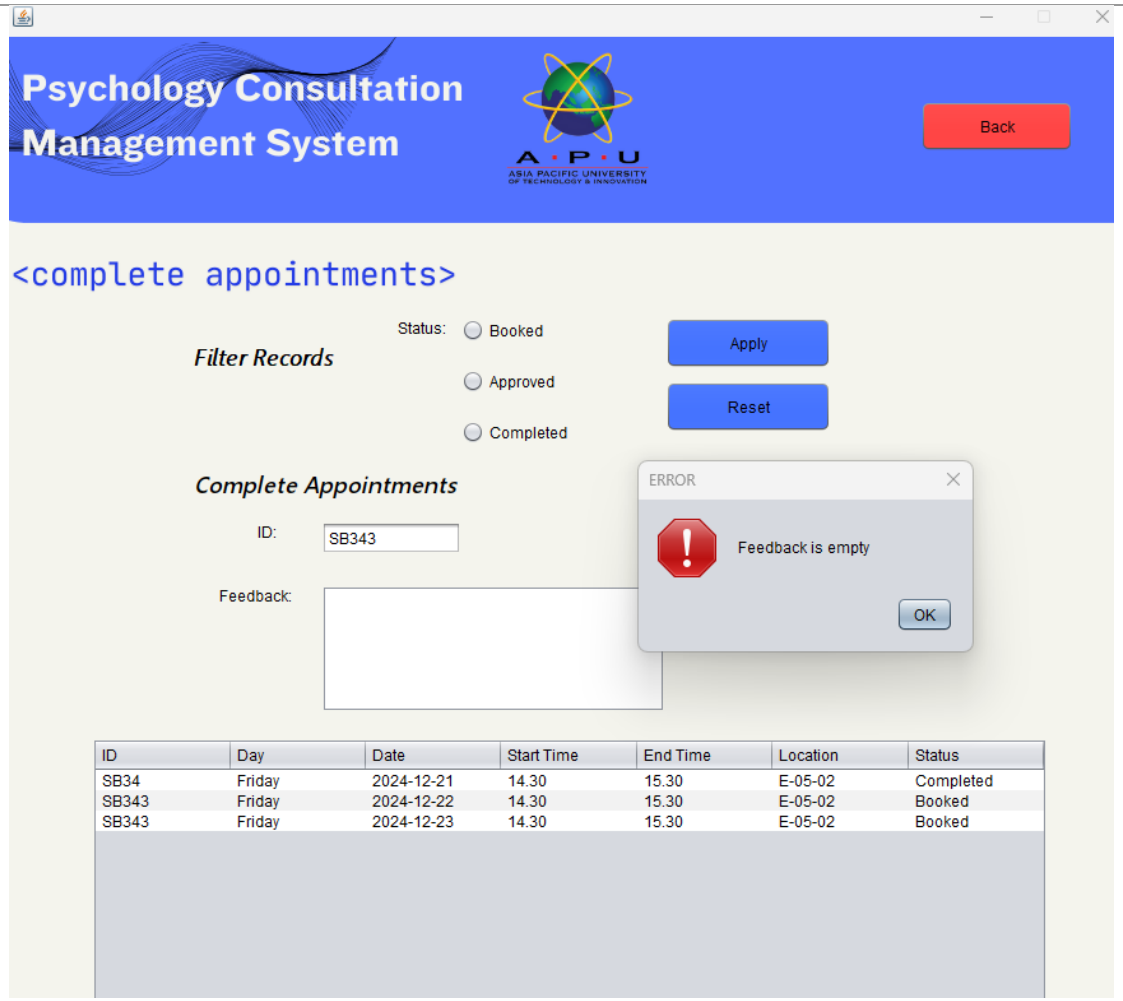


	`CN253:Saturday:2024-12-21:18.00:19.00:E-05-01`
Update Consultation	If a lecturer wishes to update the consultation slot, they can just modify the values in the table and click on update, the same validation check occurs as when a new consultation is set. If validation is successful, the record in the table will be overwritten with the new value as well as in consultation.txt 

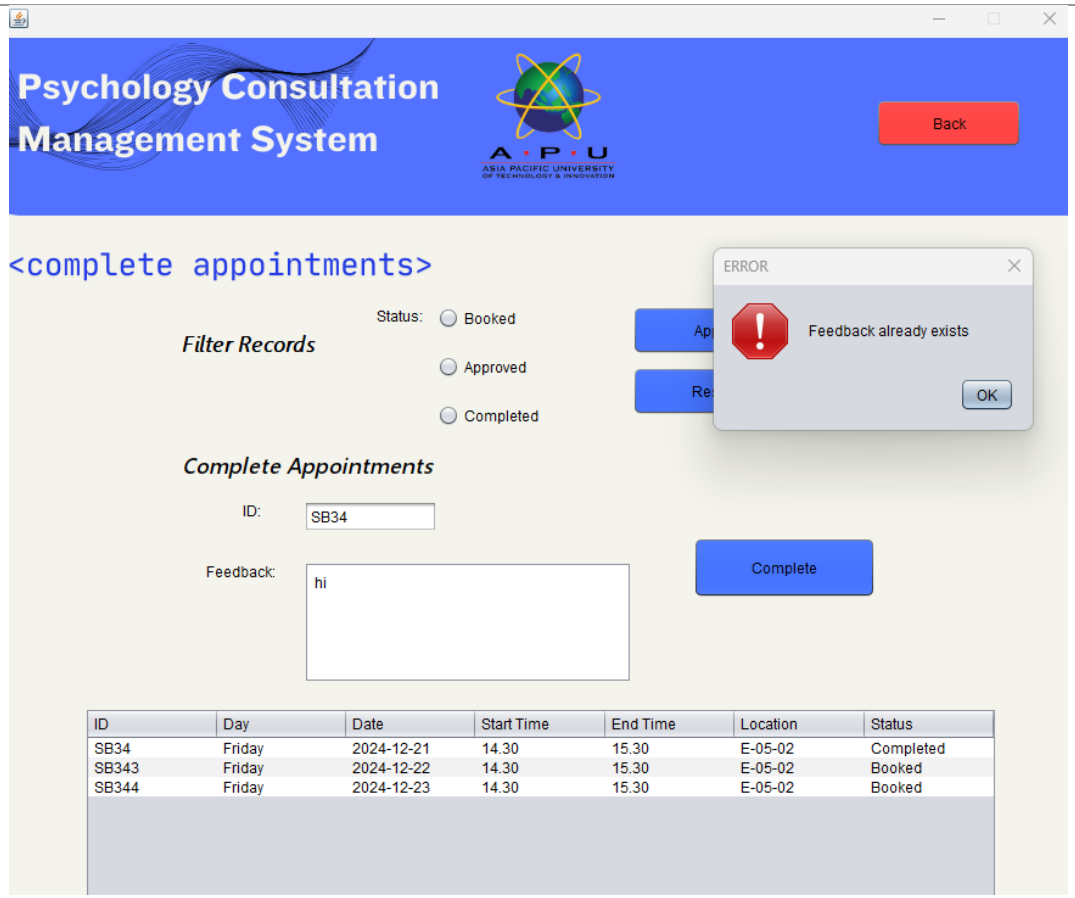
	`CN253:Saturday:2024-12-22:18.00:19.00:E-05-01`
Delete Consultation Slot	To delete a consultation slot, lecturers will have to select the Yes radio option at 'Delete Record'. Doing so will display a new text field where they must input the consultationID of the slot they wish to delete. If the ID is valid, the slot will be deleted from the table and the txt file, if not, an error prompt will appear. Both of this can be seen below 

	
Complete Appointment	If a lecturer wishes to complete an appointment, they must input the bookingID of the slot they wish to complete as well as submit feedback to the student. Validation occurs in which the system will check if the record is existent as well as if the feedback text area has a value. This can be seen in the images below.

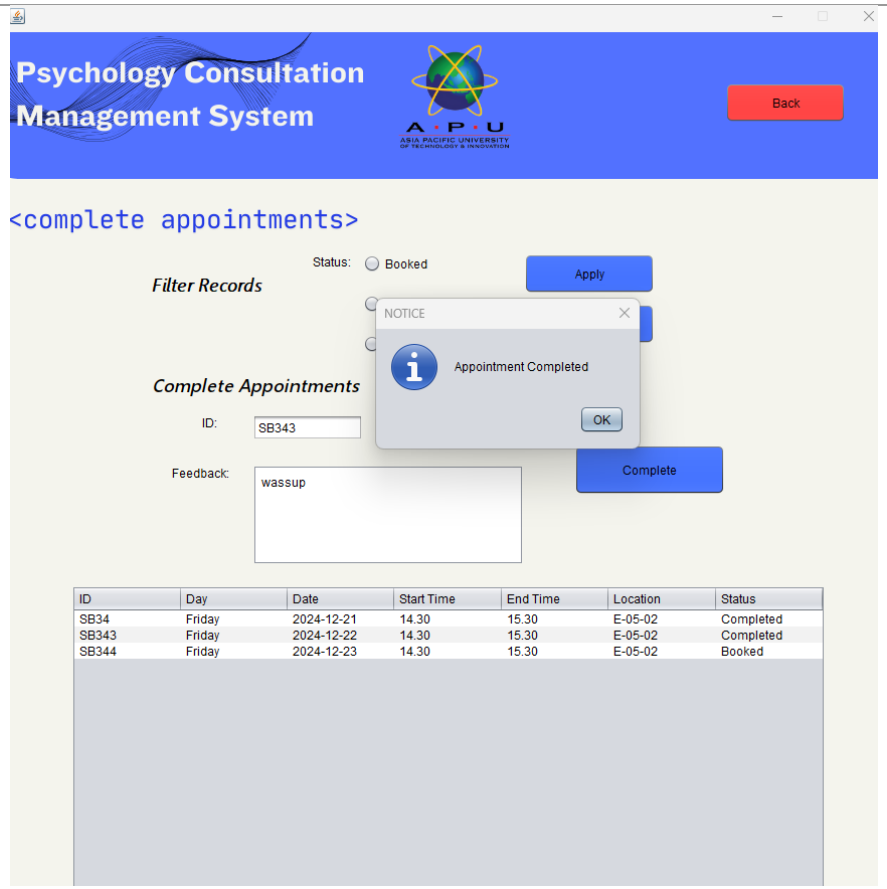

Psychology Consultation Management System
A·P·U
ASIA PACIFIC UNIVERSITY OF TECHNOLOGY & INNOVATION
Back
<complete appointments>
Filter Records
Status:
Booked
Approved
Completed
Apply
Reset
Complete Appointments
ID:
SB777
Feedback:
yo
ERROR
ID not found
OK
ID | Day | Date | Start Time | End Time | Location | Status
SB34 | Friday | 2024-12-21 | 14.30 | 15.30 | E-05-02 | Completed
SB343 | Friday | 2024-12-22 | 14.30 | 15.30 | E-05-02 | Booked
SB343 | Friday | 2024-12-23 | 14.30 | 15.30 | E-05-02 | Booked



It is important to note that records with status of ‘Completed’ cannot be completed once again. If the system detects that the lecturer is attempting to do so, an error prompt will appear seen below

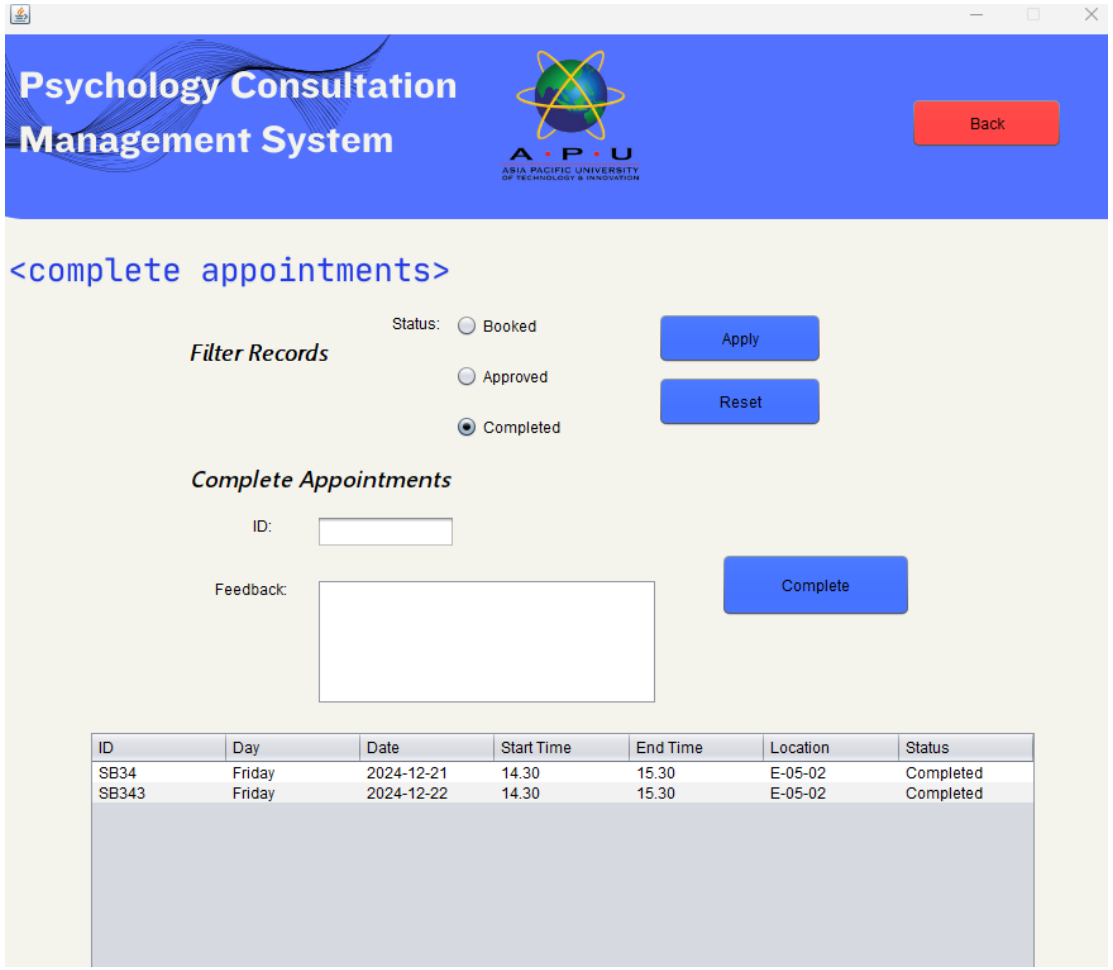


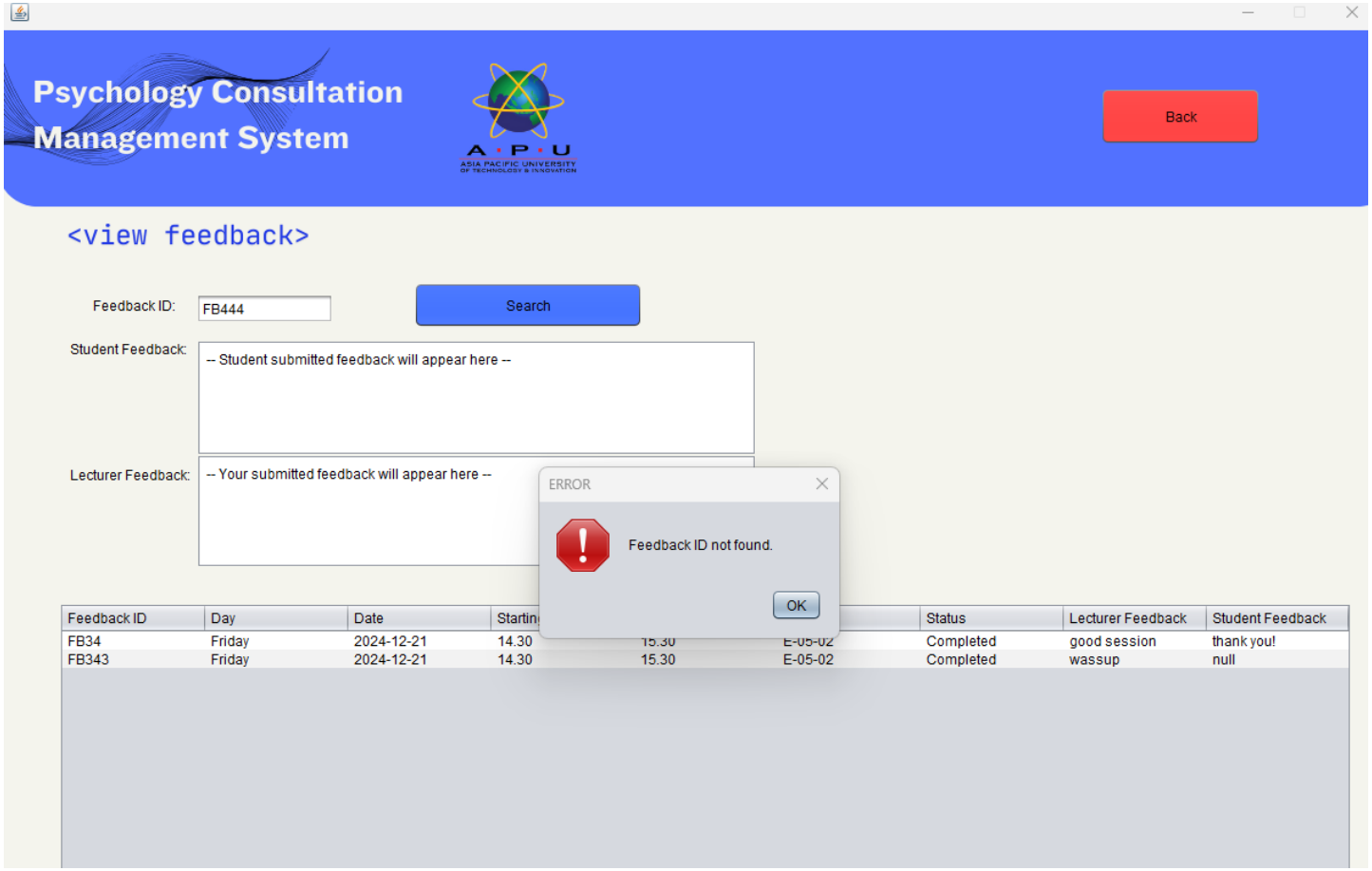
If validation is a success, the status of the selected slot will be changed to ‘Completed’ seen below as well as a notification alerting lecturers of its success. A new record in feedback.txt will also be completed which contains all the booking details as well as the lecturers feedback (student feedback is set to null since student has yet to submit one)



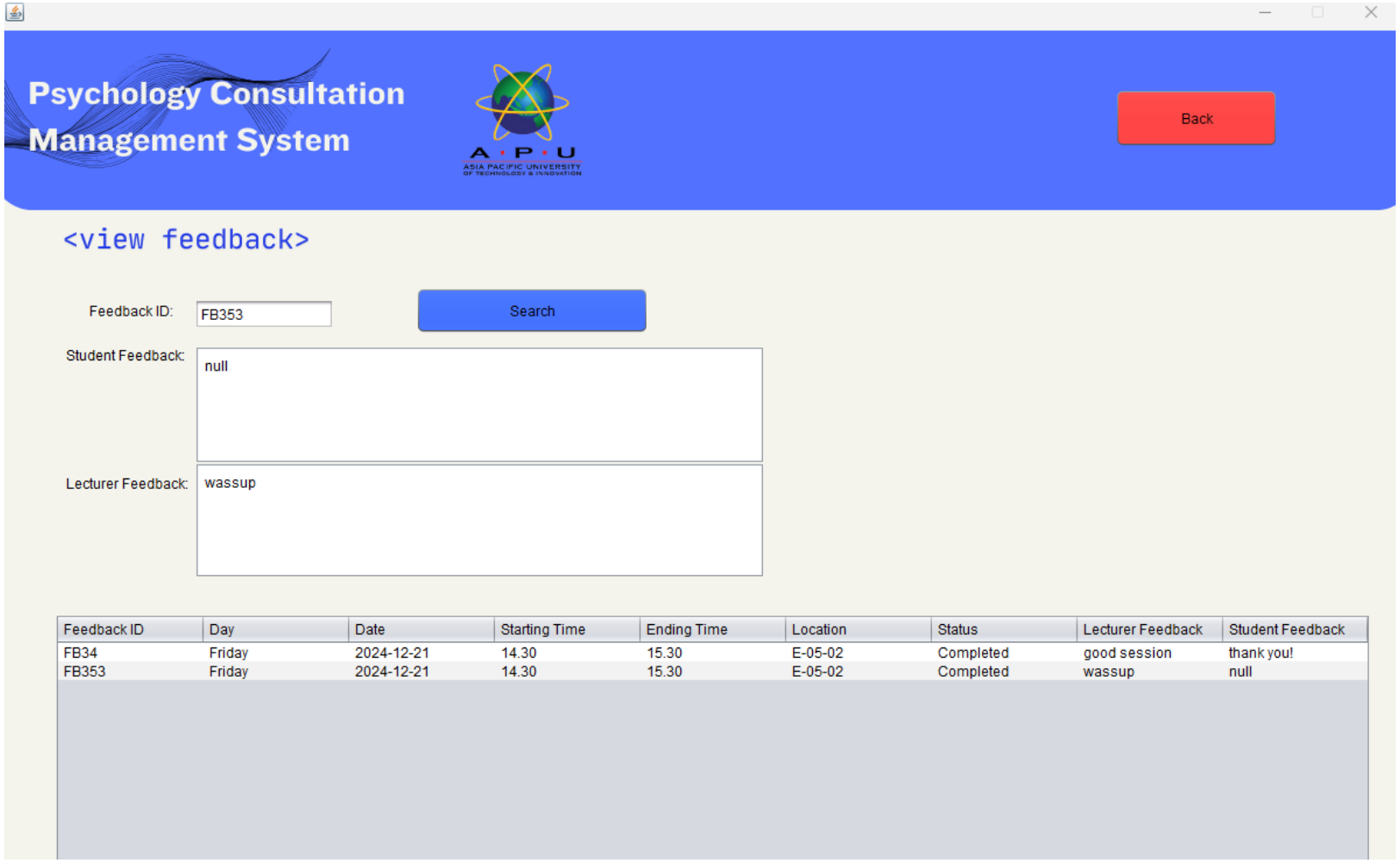

```
FB34:Friday:2024-12-21:14.30:15.30:E-05-02:Completed:good session:thank you!
FB343:Friday:2024-12-21:14.30:15.30:E-05-02:Completed:wassup:null
```

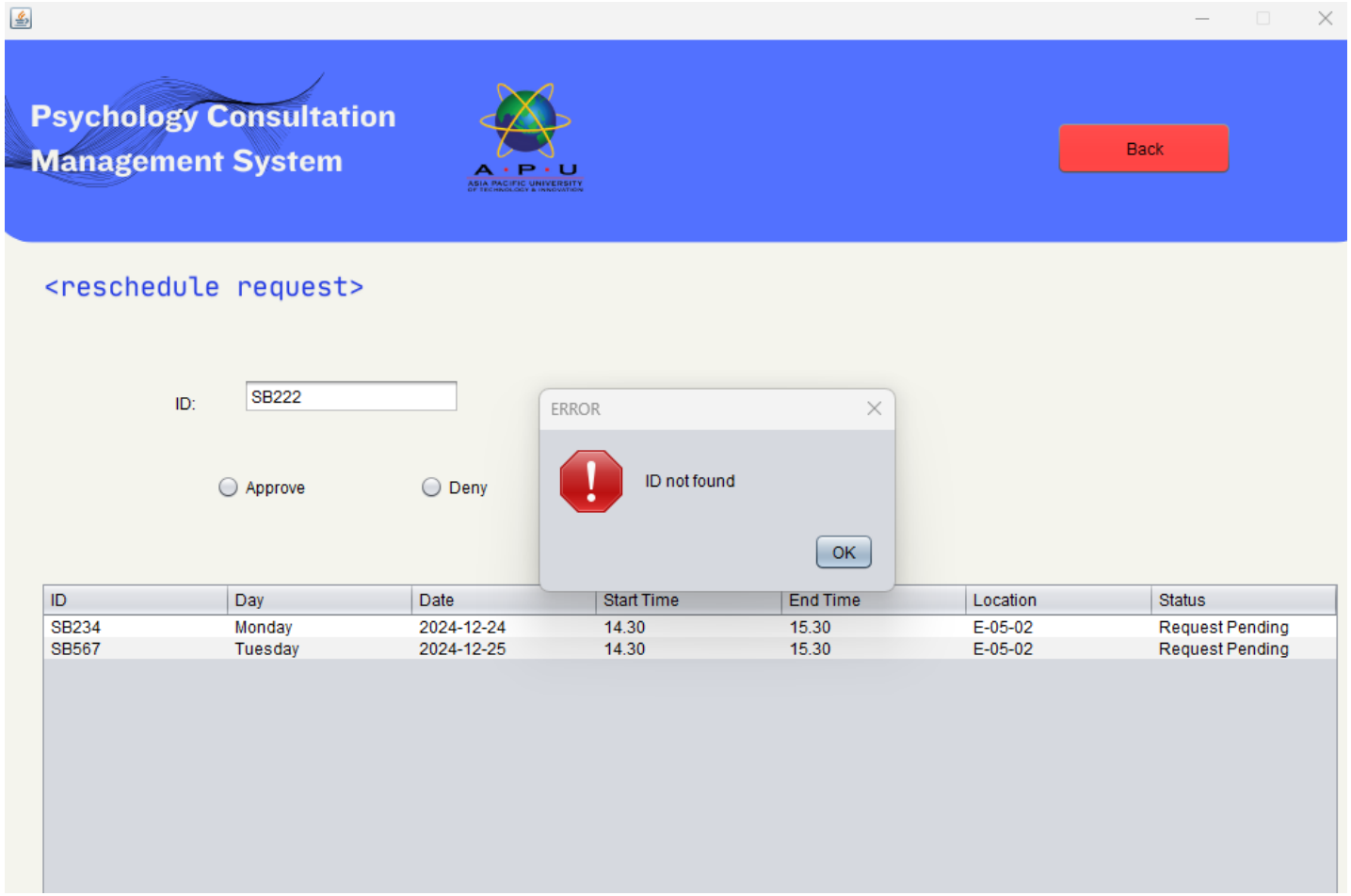
Filtering Records	Lecturers can also filter the records based on its status value. Through this, lecturers can specifically look at appointments that has already past or are upcoming. This is seen below where the lecturer is filtering the records to only display completed appointments.



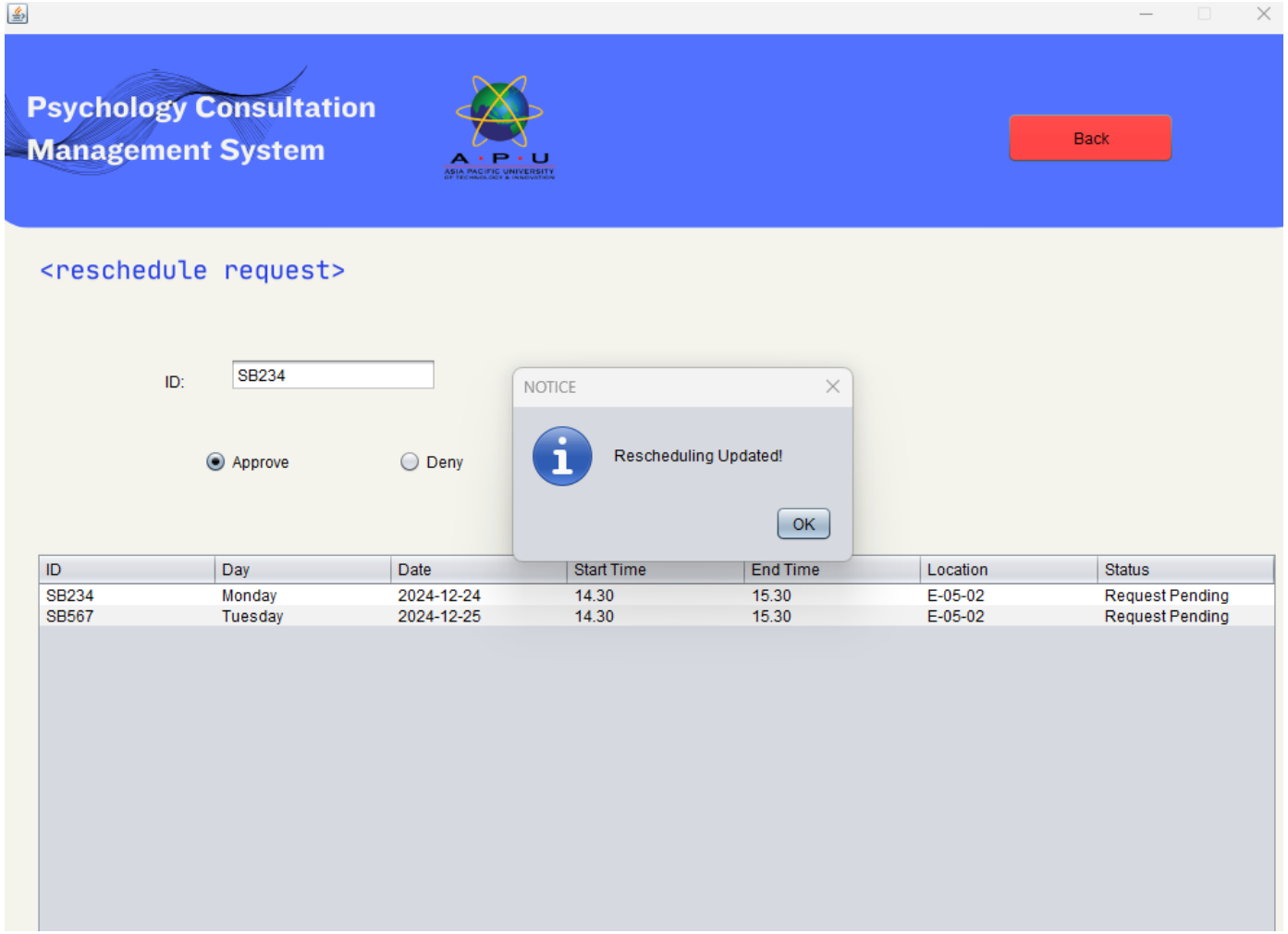
View Feedback (Lecturer)	All student feedbacks will be displayed in a table in the View Feedback section. If a lecturer wishes to view a specific one, they must enter the feedbackID of the record they wish to display. Validation will occur to ensure that the feedbackID inputted actually exists. This can be seen below. 

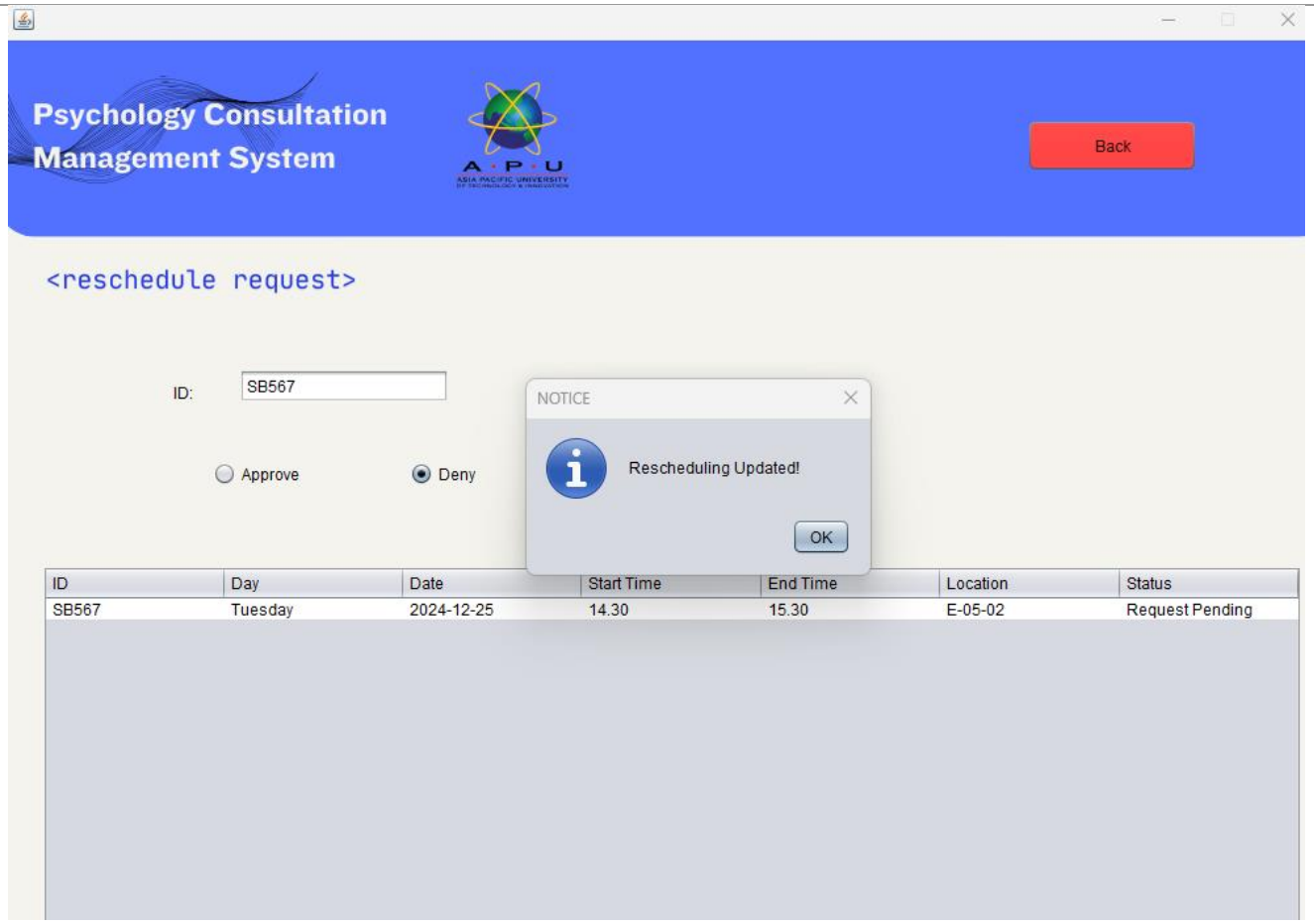
If the feedbackID exists, then the respective text area's value will be filled with the corresponding feedbacks seen below.



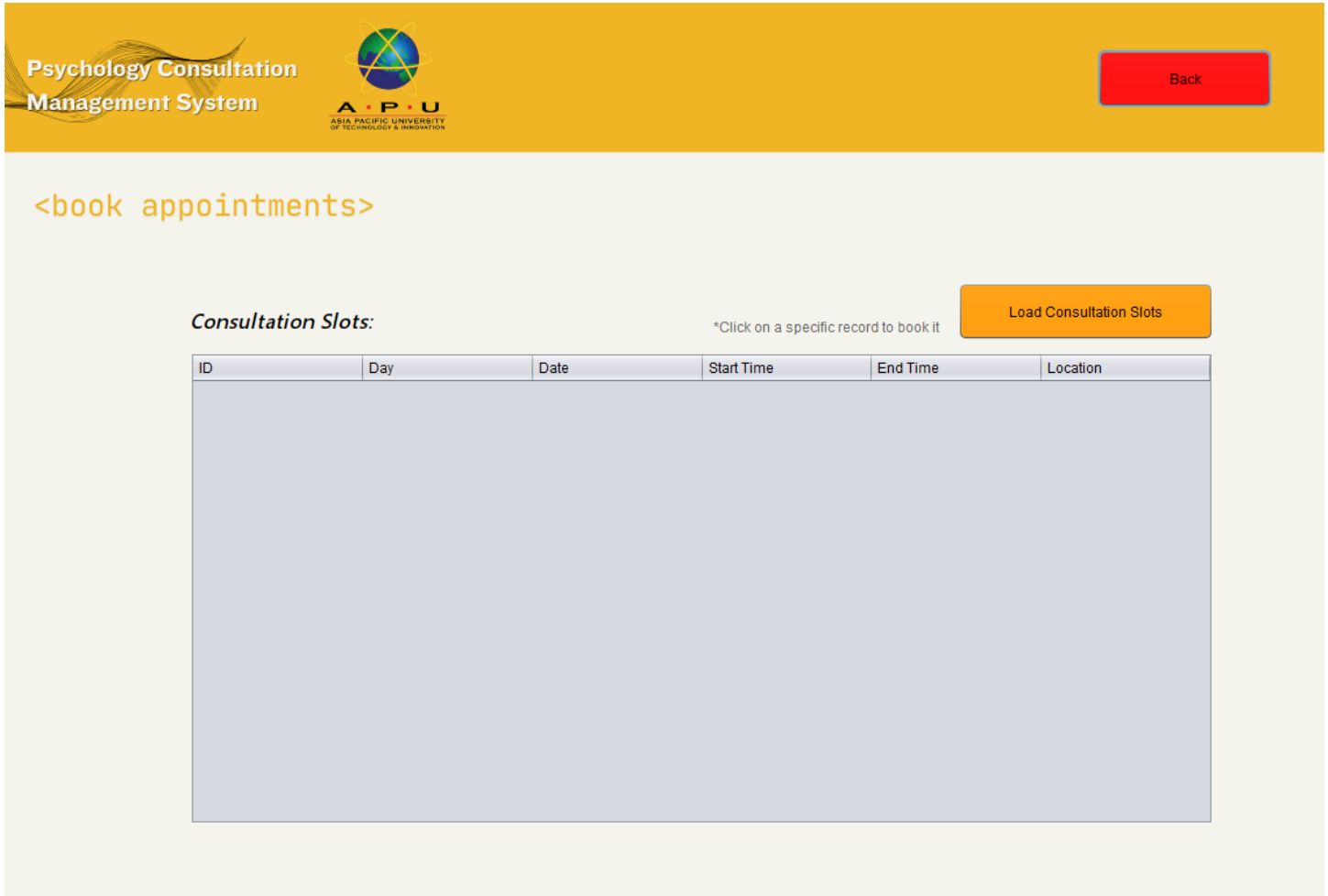
Approve / Deny Rescheduling Request	To approve or deny a rescheduling request, the lecturer will have to input the bookingID of the record they wish to update. Like other screens, validation will occur to ensure the ID is existent. If its not, an error message will appear seen below. 

If the ID does exist, the lecturer will have the option to either approve or deny the request. Doing so gets rid of the record from the table as well as change the status value of the record to ‘Approved’ and ‘Denied’ in booking.txt. This can be seen in the images below

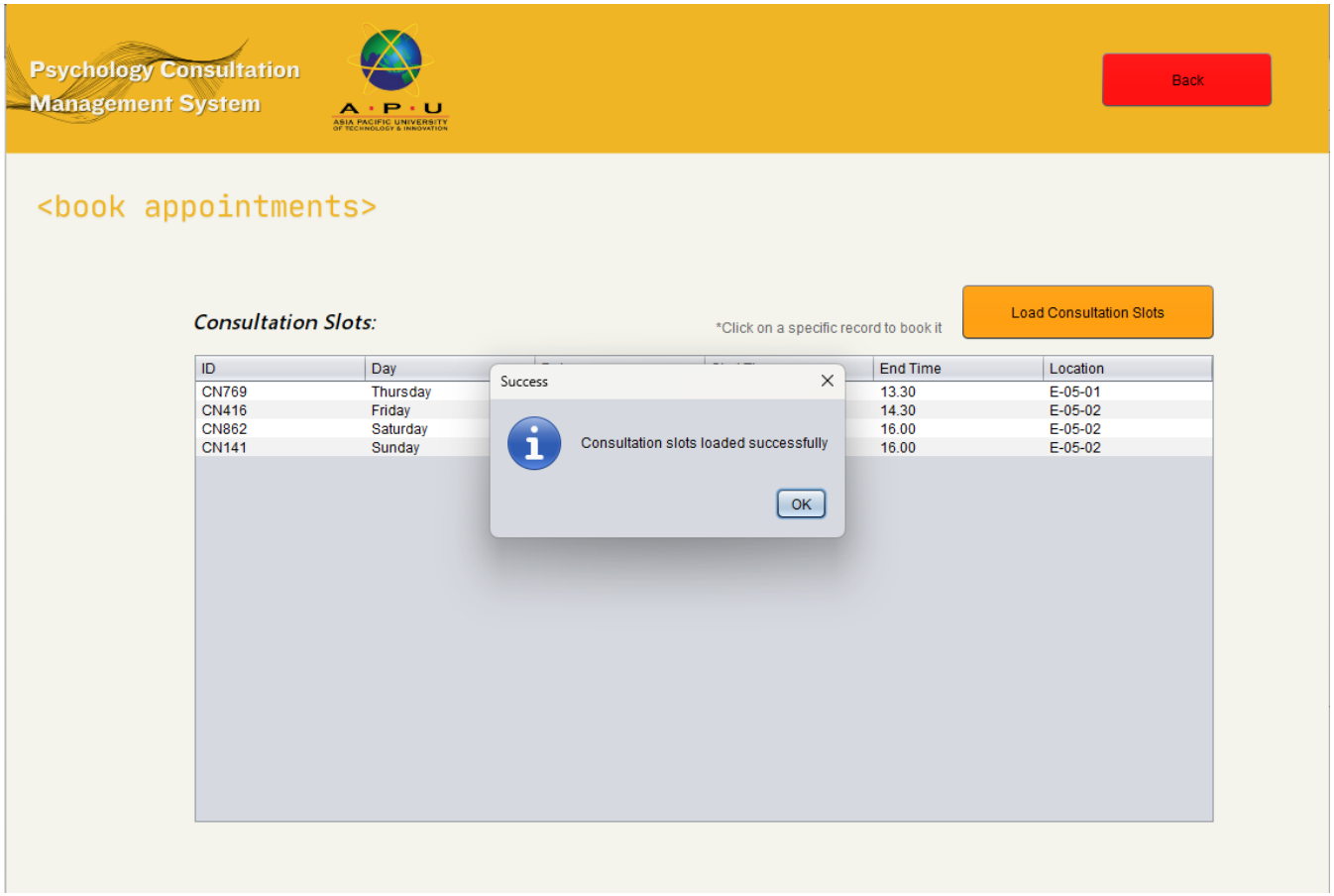




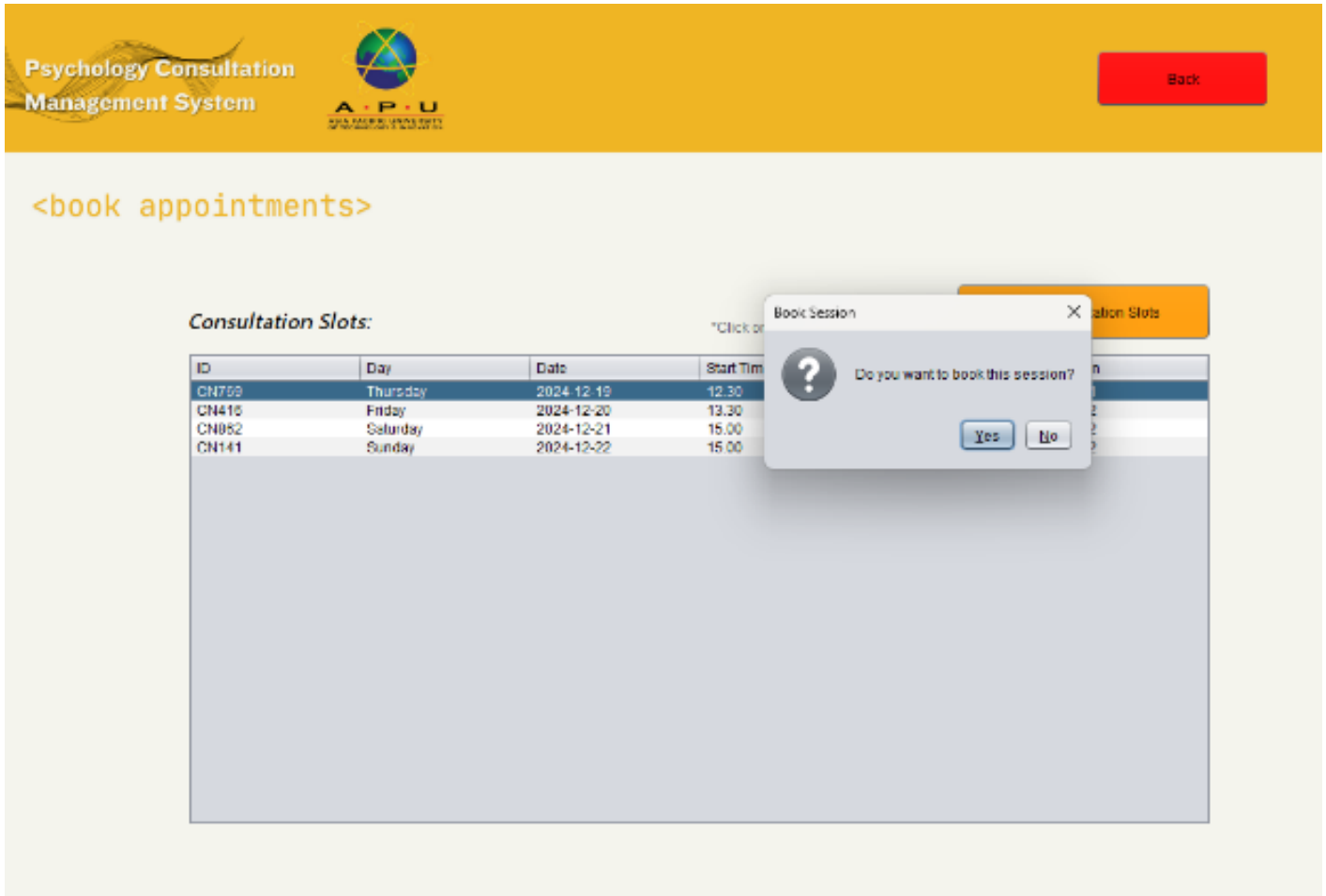

```
SB34:Friday:2024-12-21:14.30:15.30:E-05-02:Completed
SB343:Saturday:2024-12-22:14.30:15.30:E-05-02:Completed
SB344:Sunday:2024-12-23:14.30:15.30:E-05-02:Booked
SB234:Monday:2024-12-24:14.30:15.30:E-05-02:Approved
SB567:Tuesday:2024-12-25:14.30:15.30:E-05-02:Denied
```

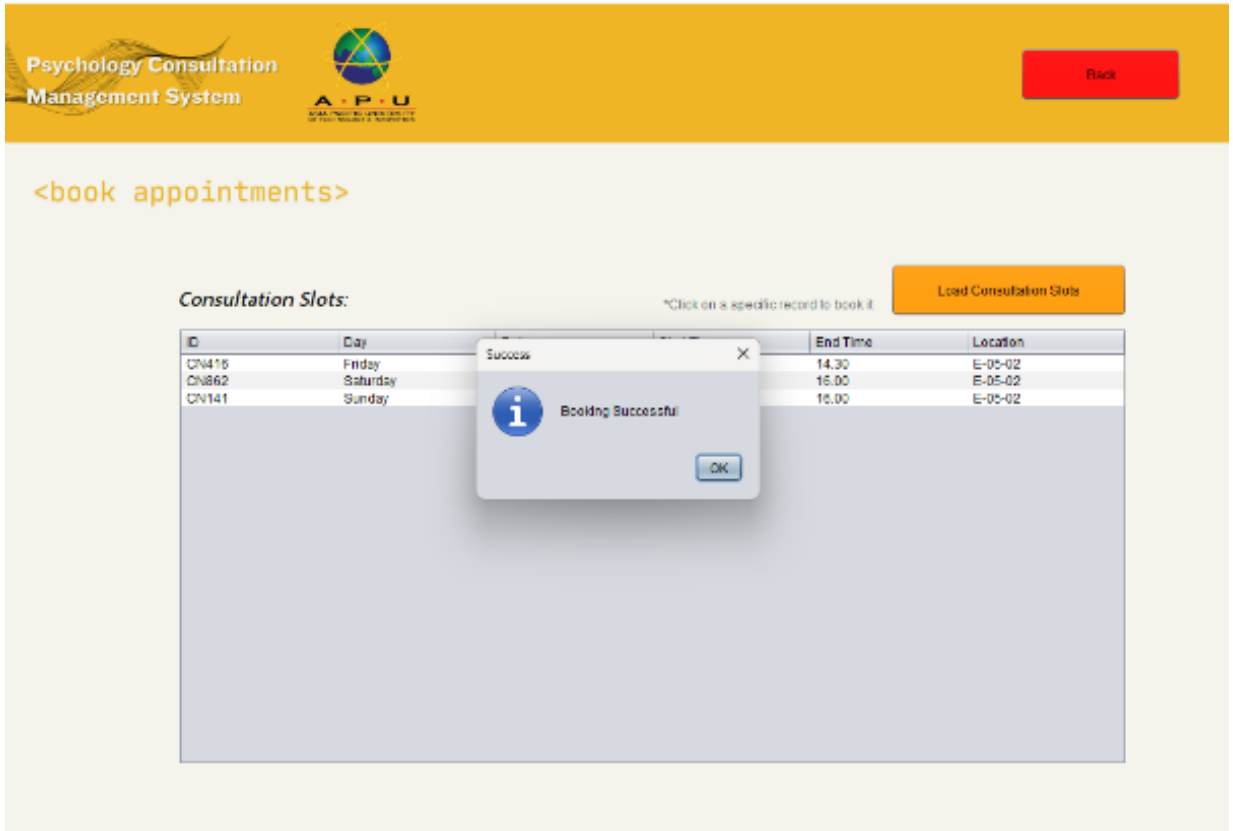
Make Booking	Students that want to view the consultation slots can click on the "Load Consultation Slots" and it will be ready for booking. 

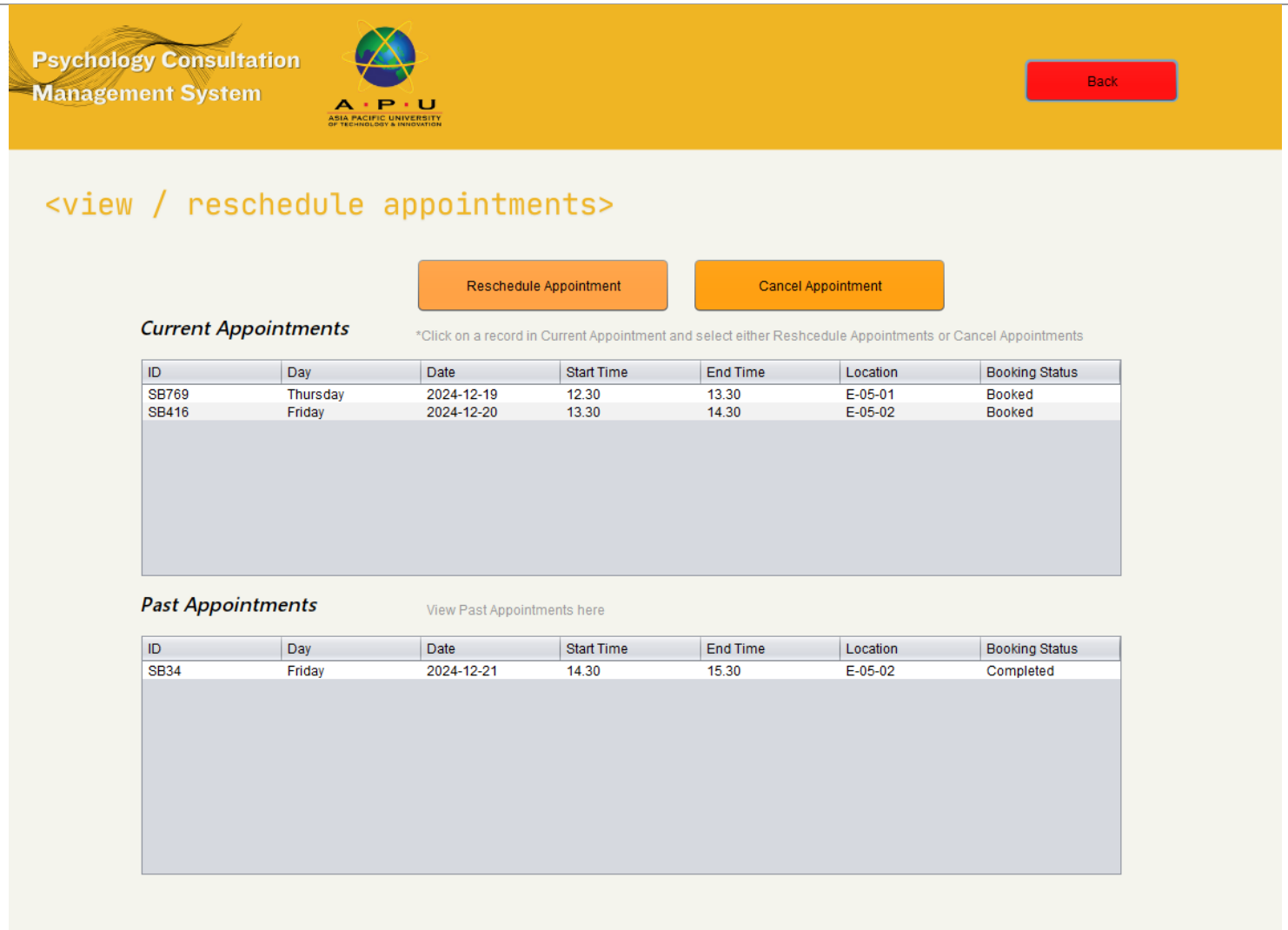
After clicking on “Load Consultation Slots”, the consultation details will appear in the table and a pop-up message will be displayed to notify students.



After clicking OK, once student click on the consultation slot, a confirmation message will appear, if students click “Yes”, a message will appear, and the consultation slot will be deleted from the table.

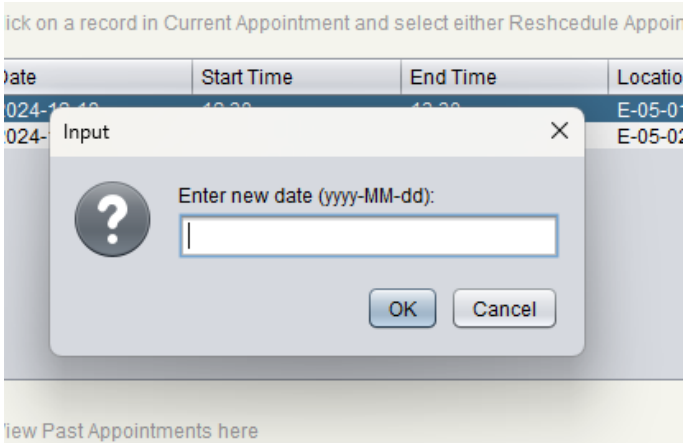


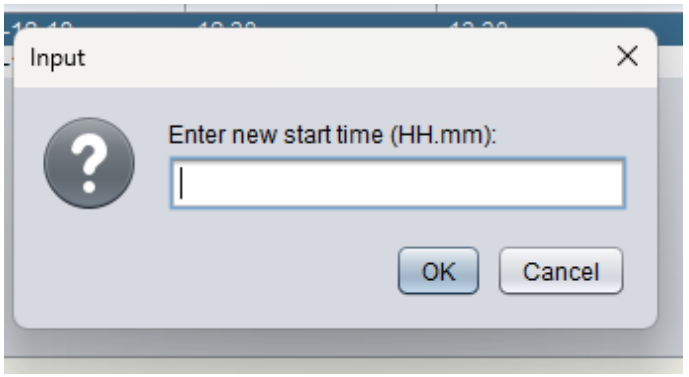
	Then, “Booking Successful” message will appear to indicate a booked consultation. 
View / Reschedule Appointments	Students can view two tables of current and past appointments as their appointments will be updated automatically. When students click on the current appointment they have booked, they can select choose to either ‘Reschedule Appointment’ or ‘Cancel Appointment’

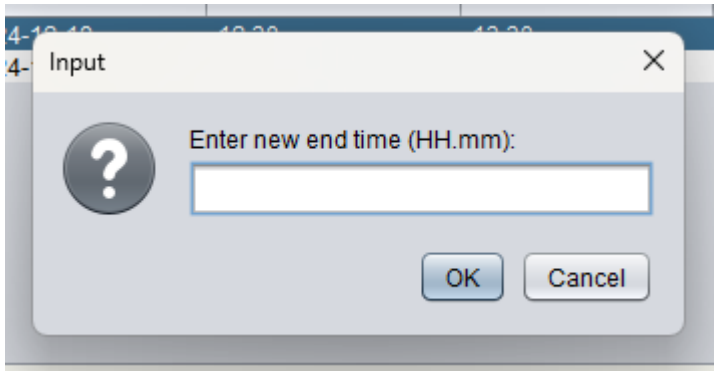

Psychology Consultation Management System
A·P·U
ASIA PACIFIC UNIVERSITY OF TECHNOLOGY & INNOVATION
Back
<view / reschedule appointments>
Reschedule Appointment
Cancel Appointment
Current Appointments
*Click on a record in Current Appointment and select either Reshcedule Appointments or Cancel Appointments
ID | Day | Date | Start Time | End Time | Location | Booking Status
SB769 | Thursday | 2024-12-19 | 12.30 | 13.30 | E-05-01 | Booked
SB416 | Friday | 2024-12-20 | 13.30 | 14.30 | E-05-02 | Booked
Past Appointments
View Past Appointments here
ID | Day | Date | Start Time | End Time | Location | Booking Status
SB34 | Friday | 2024-12-21 | 14.30 | 15.30 | E-05-02 | Completed

Option 1: ‘Reschedule Appointment’

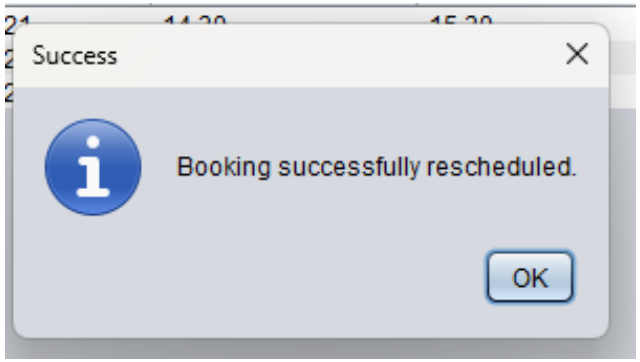
The input text field as shown below will be prompt for users to fill in.







After entering the latest details, a success message will be displayed.

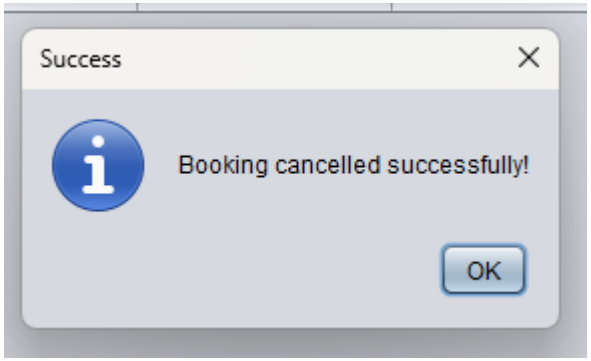


The Booking Status will be changed to ‘Request Pending’ seen below.

ID	Day	Date	Start Time	End Time	Location	Booking Status
SB34	Friday	2024-12-21	14.30	15.30	E-05-02	Completed
SB769	Thursday	2024-12-21	14.00	15.00	E-05-01	Request Pending
SB416	Friday	2024-12-20	13.30	14.30	E-05-02	Booked

Option 2: ‘Cancel Appointment’

After selecting the appointment, they would like to cancel, they will need to click on the cancel button and a message of success will be displayed.

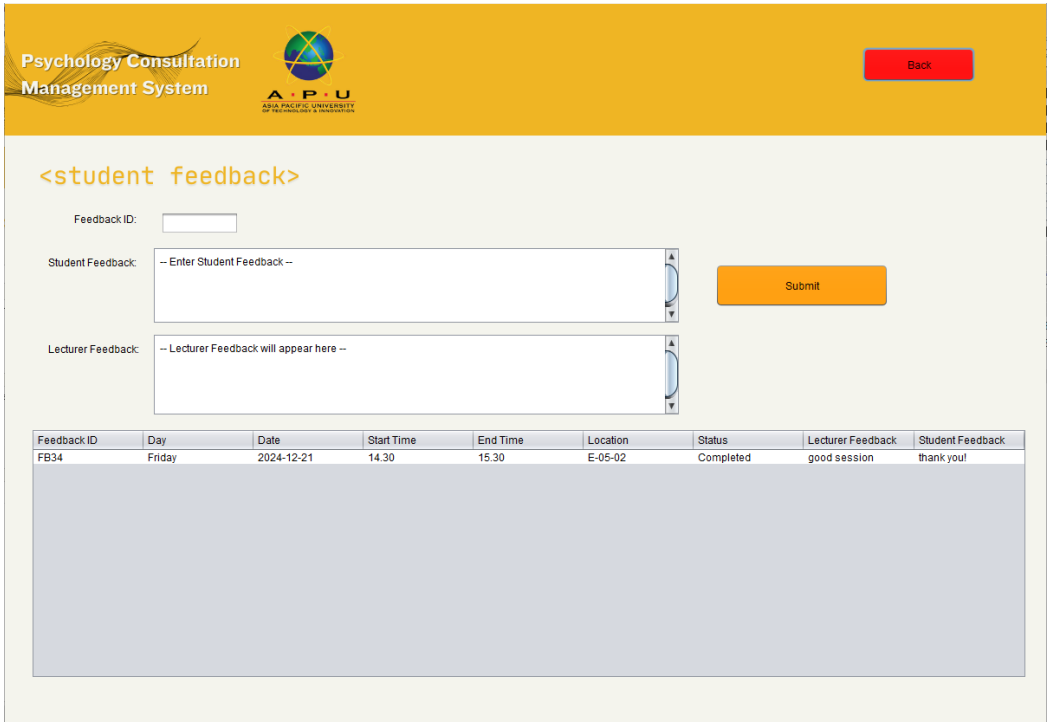


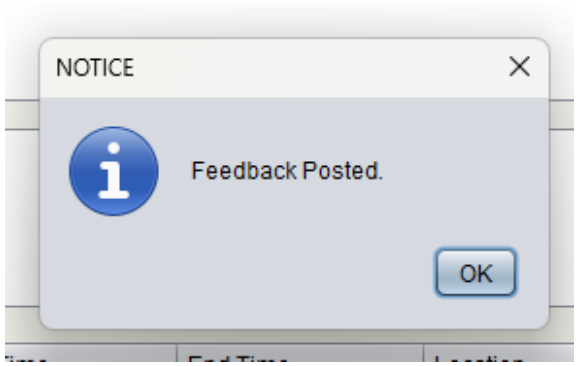
The status value of the cancelled record will now be changed to ‘Cancelled’ in booking.txt seen below

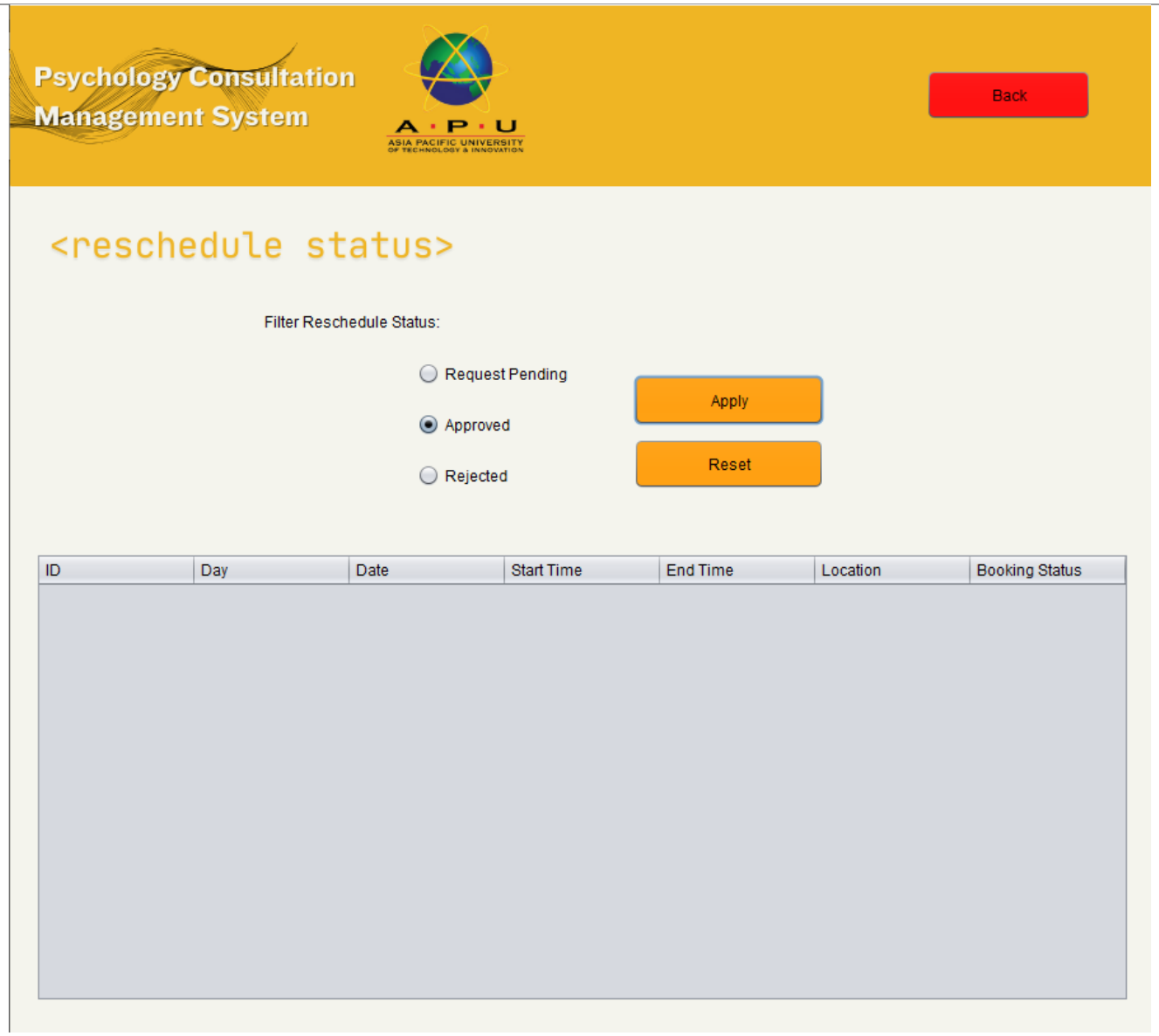
```
SB34:Friday:2024-12-21:14.30:15.30:E-05-02:Completed
SB343:Saturday:2024-12-22:14.30:15.30:E-05-02:Completed
SB344:Sunday:2024-12-23:14.30:15.30:E-05-02:Booked
SB234:Monday:2024-12-24:14.30:15.30:E-05-02:Approved
SB567:Tuesday:2024-12-25:14.30:15.30:E-05-02:Denied
SB567:Tuesday:2024-12-25:14.30:15.30:E-05-02:Cancelled
```

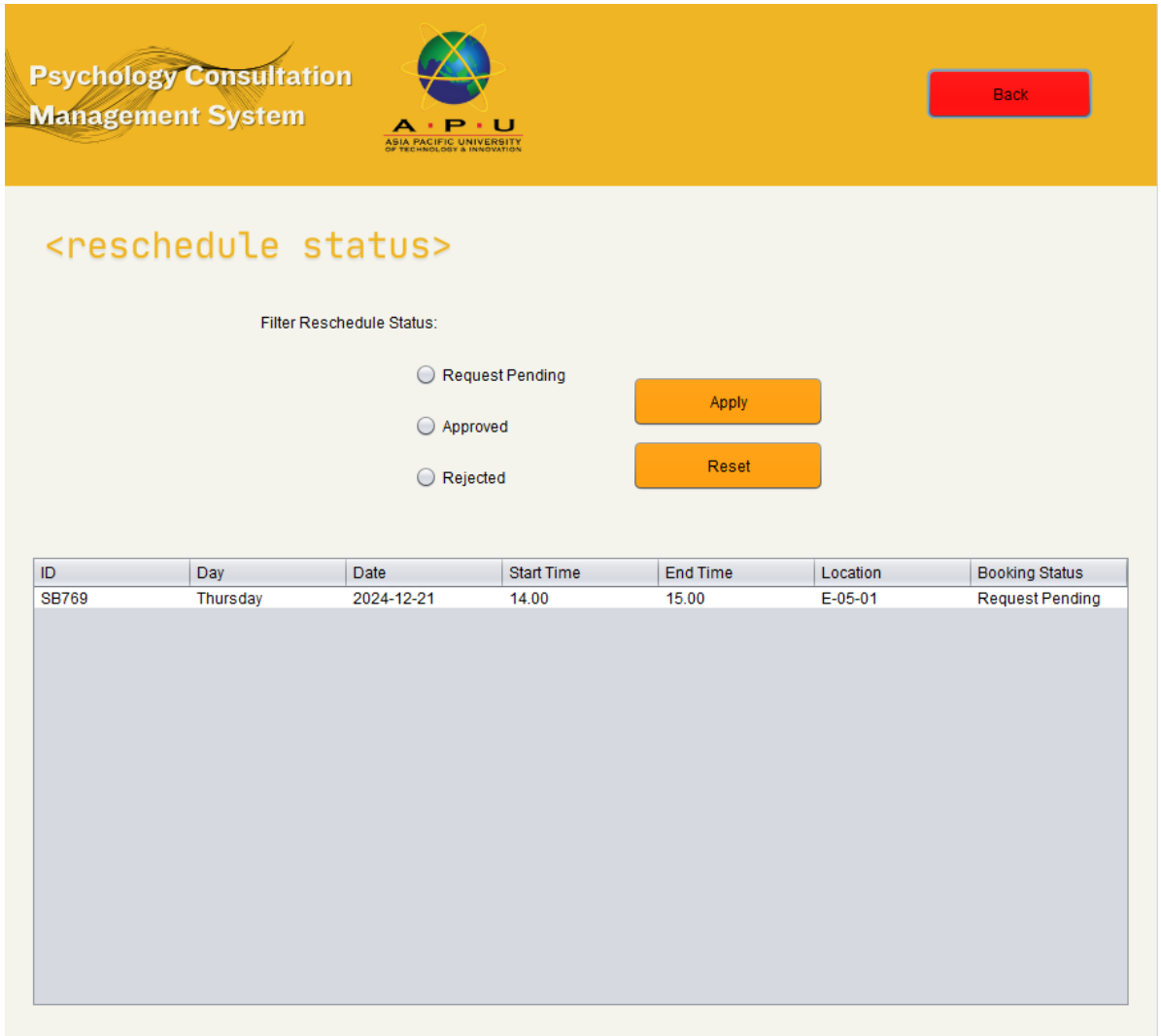
Student Feedback

Students who would like to give feedback from their consultation session can head to the feedback page as shown below:



	After students fill up their feedback in the text field and click submit, a success message will be displayed.  The feedback inputted will then be updated accordingly in 'feedback.txt' FB34:Friday:2024-12-21:14.30:15.30:E-05-02:Completed:good session:thank you! FB353:Friday:2024-12-21:14.30:15.30:E-05-02:Completed:wassup:hi!
View Reschedule Request Status	Students can view the schedule status based on the selections made such as "Request Pending", "Approved" or "Rejected"**.** Based on their chosen status they would like to view; the booking information will be loaded in the table after clicking on "Apply". Both of this can be seen in the figures below.


Psychology Consultation
Management System
A·P·U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION
Back
<reschedule status>
Filter Reschedule Status:
Request Pending
Approved
Rejected
Apply
Reset
ID
Day
Date
Start Time
End Time
Location
Booking Status


Psychology Consultation
Management System
A·P·U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION
Back
<reschedule status>
Filter Reschedule Status:
Request Pending
Approved
Rejected
Apply
Reset
ID
Day
Date
Start Time
End Time
Location
Booking Status
SB769
Thursday
2024-12-21
14.00
15.00
E-05-01
Request Pending

4. Implementation of OOP Concepts

CONSTRUCTORS / ENCAPSULATION
Booking Class

```java
public class Booking {
    private String bookID;
    private String bookDay;
    private String bookDate;
    private String bookStartTime;
    private String bookEndTime;
    private String bookLocation;
    private String bookStatus;

public Booking(String bookID, String bookDay, String bookDate, String bookStartTime, String bookEndTime, String bookLocation, String bookStatus){
    this.bookID = bookID;
    this.bookDay = bookDay;
    this.bookDate = bookDate;
    this.bookStartTime = bookStartTime;
    this.bookEndTime = bookEndTime;
    this.bookLocation = bookLocation;
    this.bookStatus = bookStatus;
}
public String getBookID(){
    return bookID;
}

public String getBookDay(){
    return bookDay;
}
public String getBookDate(){
    return bookDate;
}
public String getBookStartTime(){
    return bookStartTime;
}
public String getBookEndTime(){
    return bookEndTime;
}
public String getBookLocation(){
    return bookLocation;
}
public String getBookStatus(){
    return bookStatus;
}
```

```java
public void setConsultDate(String newDate){
    this.bookDate = newDate;
}
public void setConsultStartTime(String newStartTime){
    this.bookStartTime = newStartTime;
}
public void setConsultEndTime(String newEndTime){
    this.bookEndTime = newEndTime;
}
public void setBookStatus(String newStatus){
    this.bookStatus = newStatus;
}
public String toString(){
        return bookID + ":" + bookDay + ":" + bookDate + ":" + bookStartTime + ":" +
                bookEndTime + ":" + bookLocation + ":" + bookStatus;
    }
}
```

Constructors, which are a core fundamental of Object-Oriented Programming (OOP), were applied in our system to help define our required classes, which can be seen in the figure above. Also evident in the figure above, is the application of encapsulation in which the Booking class would group attributes such as bookID, bookDay, bookDate, etc. in a single unit. The attributes are declared as private attributes to restrict direct access from other classes. With the concept of encapsulation, this would protect the internal state of the object and only certain public methods (getters and setters) are allowed to modify the objects such as getBookID(), getBookDate() and setBookStatus(). Therefore, encapsulation is beneficial towards maintaining the integrity of the data and protecting any modification mishaps from happening. Essentially with encapsulation, we are morphing the class to feature **public** methods, with **private** attributes. This is especially useful for constructors where data integrity and security is second to none. Due to this, we also applied the concept into the following classes:

- Student Class
- Lecturer Class
- Consultation Class
- Booking Class (**see above**)
- Feedback Class

* Screenshots of the encapsulation for the other classes can be seen below

Student Class

```java
public class Student {
    private String studentID;
    private String studentName;
    private String studentPassword;
    private String studentGender;

public Student(String studentID, String Name, String Password, String Gender){
    this.studentID = studentID;
    this.studentName = Name;
    this.studentPassword = Password;
    this.studentGender = Gender;

}

public String getStudentID(){
    return studentID;
}

public String getStudentName(){
    return studentName;
}

public String getStudentPassword(){
    return studentPassword;
}

public String getStudentGender(){
    return studentGender;
}

public boolean isValidLogin(String loginUsername, String loginPassword) {
        return this.studentName.equals(loginUsername) && this.studentPassword.equals(loginPassword);
}

public String toString(){
    return studentID + ":" + studentName + ":" + studentPassword + ":" + studentGender;
}
}
```

Lecture Class

```java
10  public class Lecturer {
        private String lecturerID;
        private String lecturerName;
        private String lecturerPassword;
        private String lecturerGender;
15
16  public Lecturer (String lecturerID, String Name, String Password, String Gender){
17      this.lecturerID = lecturerID;
18      this.lecturerName = Name;
19      this.lecturerPassword = Password;
20      this.lecturerGender = Gender;
21
22  }
23
24  public String getLecturerID (){
25      return lecturerID;
26  }
27
28  public String getLecturerName(){
29      return lecturerName;
30  }
31
32  public String getLeturerPassword(){
33      return lecturerPassword;
34  }
35
36  public String lecturerGender(){
37      return lecturerGender;
38  }
39
40   public boolean isValidLogin(String loginUsername, String loginPassword) {
41          return this.lecturerName.equals(loginUsername) && this.lecturerPassword.equals(loginPassword);
42      }
43
44  public String toString(){
45      return lecturerID + ":" + lecturerName + ":" + lecturerPassword + ":" + lecturerGender;
46  }
47  }
```

Consultation Class

```java
public class ConsultationSlot {
    private String consultID;
    private String consultDay;
    private String consultDate;
    private String consultStartTime;
    private String consultEndTime;
    private String consultLocation;

public ConsultationSlot(String consultID, String consultDay, String consultDate, String consultStartTime, String consultEndTime, String consultLocation){
    this.consultID = consultID;
    this.consultDay = consultDay;
    this.consultDate = consultDate;
    this.consultStartTime = consultStartTime;
    this.consultEndTime = consultEndTime;
    this.consultLocation = consultLocation;
}

public String getConsultID(){
    return consultID;
}

public String getConsultDay(){
    return consultDay;
}

public String getConsultDate(){
    return consultDate;
}

public String getConsultStartTime(){
    return consultStartTime;
}

public String getConsultEndTime(){
    return consultEndTime;
}

public String getConsultLocation(){
    return consultLocation;
}

    @Override
    public String toString(){
    return consultID + ":" + consultDay + ":" + consultDate + ":" + consultStartTime
            + ":" + consultEndTime + ":" +consultLocation;
}
```

Feedback Class

```java
public class Feedback extends Booking {
    private String feedbackID;
    private String feedbackLecturer;
    private String feedbackStudent;

public Feedback (String feedbackID, String bookDay, String bookDate, String bookStartTime, String bookEndTime, String bookLocation, String bookStatus,
        String feedbackLecturer, String feedbackStudent) {

super("bookID", bookDay, bookDate, bookStartTime, bookEndTime, bookLocation, bookStatus);

this.feedbackID = feedbackID;
this.feedbackLecturer = feedbackLecturer;
this.feedbackStudent = feedbackStudent;
}

public String getFeedbackID(){
    return feedbackID;
}

public String getFeedbackLecturer(){
    return feedbackLecturer;
}

public String getFeedbackStudent(){
    return feedbackStudent;
}

public void setFeedbackStudent(String completeStudentFeedback){
    this.feedbackStudent = completeStudentFeedback;
}

    @Override
    public String toString(){
    return feedbackID + ":" + getBookDay() + ":" + getBookDate() + ":" + getBookStartTime()
            + ":" + getBookEndTime() + ":" + getBookLocation() + ":" + getBookStatus() +
            ":" + feedbackLecturer + ":" + feedbackStudent;
}
}
```

It is worth noting that Feedback class also applies the OOP concept of Inheritance where it inherits the values ‘bookDay, bookStartTime, bookEndTime, bookLocation & bookStatus.

OBJECT CREATION / ARRAY OF OBJECTS

As mentioned previously, constructors are used to define our class, as in they help define its attributes and methods that make said class; however, that's all they do, in order to actually utilise them, we would have to create an object / an array of object of said class. This very concept is the very foundation of OOP, therefore its utilisation in our program is way too numerous to showcase all so only a select few examples will be showcased below with each accompanied with a brief caption to capture the context of the code.

```
BookingFileHandler files = new BookingFileHandler();
try {
    if (files.doesRecordExists(completeID) && (!completeLecturerFeedback.isEmpty())){
        files.updateRecord(model, bookID, bookDay, bookDate, bookStartTime, bookEndTime, bookLocation, bookStatus, completeID);
        loadBookingsIntoTable();
        String feedbackID = completeID.replace("SB","FB");
        if(!feedback.doesRecordExists(feedbackID)){
            feedback.writeRecord(model, row, completeLecturerFeedback, completeStudentFeedback, feedbackID);
            JOptionPane.showMessageDialog(null, "Appointment Completed", "NOTICE", JOptionPane.INFORMATION_MESSAGE);

        } else {
            JOptionPane.showMessageDialog(null, "Feedback already exists", "ERROR", JOptionPane.ERROR_MESSAGE);
        }


    } else if (!files.doesRecordExists(completeID)){
        JOptionPane.showMessageDialog(null, "ID not found", "ERROR", JOptionPane.ERROR_MESSAGE);
    }
```

The code above showcases the BookingFileHandler class be created into an object which is referenced by 'files'. From this, the methods under BookingFileHandler (doesRecordExists, updateRecord) are utilised by parsing the values that satisfies the method's parameters. In this case, doesRecordExists serves to check whether the input (completeID) exists in our booking.txt text file. This is to assure that the record we are tying to alter is existent. If that is the case (doesRecordExists returns true) the program will proceed to utilise the updateRecord method to overwrite the existing record in booking.txt with the new values. If not (doesRecordExists returns false), the program will display an alert window notifying the user that the record is non-existent.

```
BookingFileHandler bookingFileHandler = new BookingFileHandler();
List<Booking> bookings = bookingFileHandler.fetchBookingData();

DefaultTableModel model = (DefaultTableModel) tAppointment.getModel();
model.setRowCount(0);
for (Booking booking : bookings) {
if ((booking.getBookStatus().equals("Completed")) || (booking.getBookStatus().equals("Booked"))
        || (booking.getBookStatus().equals("Approved"))) {
    model.addRow(new Object[]{
        booking.getBookID(),
        booking.getBookDay(),
        booking.getBookDate(),
        booking.getBookStartTime(),
        booking.getBookEndTime(),
        booking.getBookLocation(),
        booking.getBookStatus()
    });
}
}
```

This code however creates an object (list array of Booking class) which is then parsed through the fetchBookingData method found in bookingFileHandler class. In this case, the code above is utilised to obtain a list array of all records that exists in booking.txt with the assistance of the getter methods defined in Booking class to help us obtain the actual values. Once this is done, the values are then displayed in tAppointment table only if the Status value is equal to ‘Completed’ / ‘Booked’ / ‘Approved’.

INTERFACE / ABSTRACTION

Interfaces in java acts a sort of mechanism to help achieve abstraction. To further elaborate, interfaces is a completely abstract class that are used to house abstract methods which are methods which are structurally defined but contain empty bodies. Interfaces are utilised to help enable multiple inheritance as well as facilitate polymorphism. Abstraction on the other hand, is generally used to enforce upon subclasses to adhere

towards the structure of the interface by the intricate details of implementation allowing them to solely focus on the high-level functionality thus assuring optimisation and consistency. In our case an interface called FileHandler (**seen below**) was created which houses the abstract methods writeRecord, deleteRecord, readRecord, doesRecordExists & updateRecord. This interface was created to help streamline the coding process of functionality that involves any sort of file handling / manipulation.

```
interface FileHandler {
    public boolean doesRecordExists();
    public void readRecord();
    public void writeRecord();
    public void deleteRecord();
    public void updateRecord();
}
```

POLYMORPHISM

As mentioned previously, an interface also facilitates polymorphism in which multiple methods of the same operation can behave differently. The implementation of this can be seen in the utilisation of the abstract methods in many of our classes, particularly those that handle file manipulation. An example of which can be seen in the figures below where both LecturerFileHandler and ConsultationFileHandler class employs the abstract method writeRecord. In LecturerFileHandler, writeRecord is used to help in write the lecturer record in lecturer.txt by parsing the Lecturer object; in ConsultationFileHanlder however, the same method(writeRecord) is used to do the exact same function, but with the Consultation object instead.

```
public class ConsultationFileHandler implements FileHandler {
    private static final String filename = "consultation.txt";

public void writeRecord(ConsultationSlot consultation) throws IOException {
    try(BufferedWriter bw = new BufferedWriter(new FileWriter(filename, true))){
        bw.write(consultation.toString());
        bw.newLine();
    }
}
```

```java
public class LecturerFileHandler implements FileHandler {
    private static final String filename = "lecturer.txt";

    public boolean doesRecordExists (Lecturer lecturer) throws FileNotFoundException, IOException{
        try (BufferedReader br = new BufferedReader(new FileReader(filename))){
            String line;
            while ((line = br.readLine()) != null){
                String [] record = line.split(":");
                String existingLecturer = record[1];
                if (existingLecturer.equals(lecturer.getLecturerName())){
                    return true;
                }
            }
        }
        return false;
    }

    public void writeRecord(Lecturer lecturer) throws IOException{
        try(BufferedWriter bw = new BufferedWriter(new FileWriter(filename, true))){
            bw.write(lecturer.toString());
            bw.newLine();
        }
    }

    @Override
    public boolean doesRecordExists() {
        throw new UnsupportedOperationException("Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public void readRecord() {
        throw new UnsupportedOperationException("Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public void writeRecord() {
        throw new UnsupportedOperationException("Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }
```

Beyond the example above, many such instances of polymorphism can be found in our code in the classes listed below alongside the abstract methods they utilise:

- LecturerFileHandler (writeRecord, doesRecordExists)
- ConsultationFileHandler (writeRecord, updateRecord, deleteRecord)
- StudentFileHandler (writeRecord, doesRecordExists)

- FeedbackFileHandler (writeRecord, doesRecordExists)
- BookingFileHandler (writeRecord, doesRecordExists, updateRecord)

INHERITANCE

Inheritance in java entails the creation of a new class (subclass) based on an existing one (superclass) in which the subclass will be able to share and 'inherit' the super class's attributes and methods. Beyond that, inheritance is also used to establish some sort of data hierarchy. In our program inheritance can be seen in our Feedback class which acts as a subclass towards our Booking class. This is done due to the fact that we wish to store the booking details inside booking.txt inside our feedback.txt file as well; therefore, by utilising inheritance, we can simply inherit those details rather than creating a new set of the same values which will lead to an increase in redundancy and complexity. This can be seen in the figures below.

Booking Class

```java
public class Booking {
    private String bookID;
    private String bookDay;
    private String bookDate;
    private String bookStartTime;
    private String bookEndTime;
    private String bookLocation;
    private String bookStatus;

public Booking(String bookID, String bookDay, String bookDate, String bookStartTime, String bookEndTime, String bookLocation, String bookStatus){
    this.bookID = bookID;
    this.bookDay = bookDay;
    this.bookDate = bookDate;
    this.bookStartTime = bookStartTime;
    this.bookEndTime = bookEndTime;
    this.bookLocation = bookLocation;
    this.bookStatus = bookStatus;
}
```

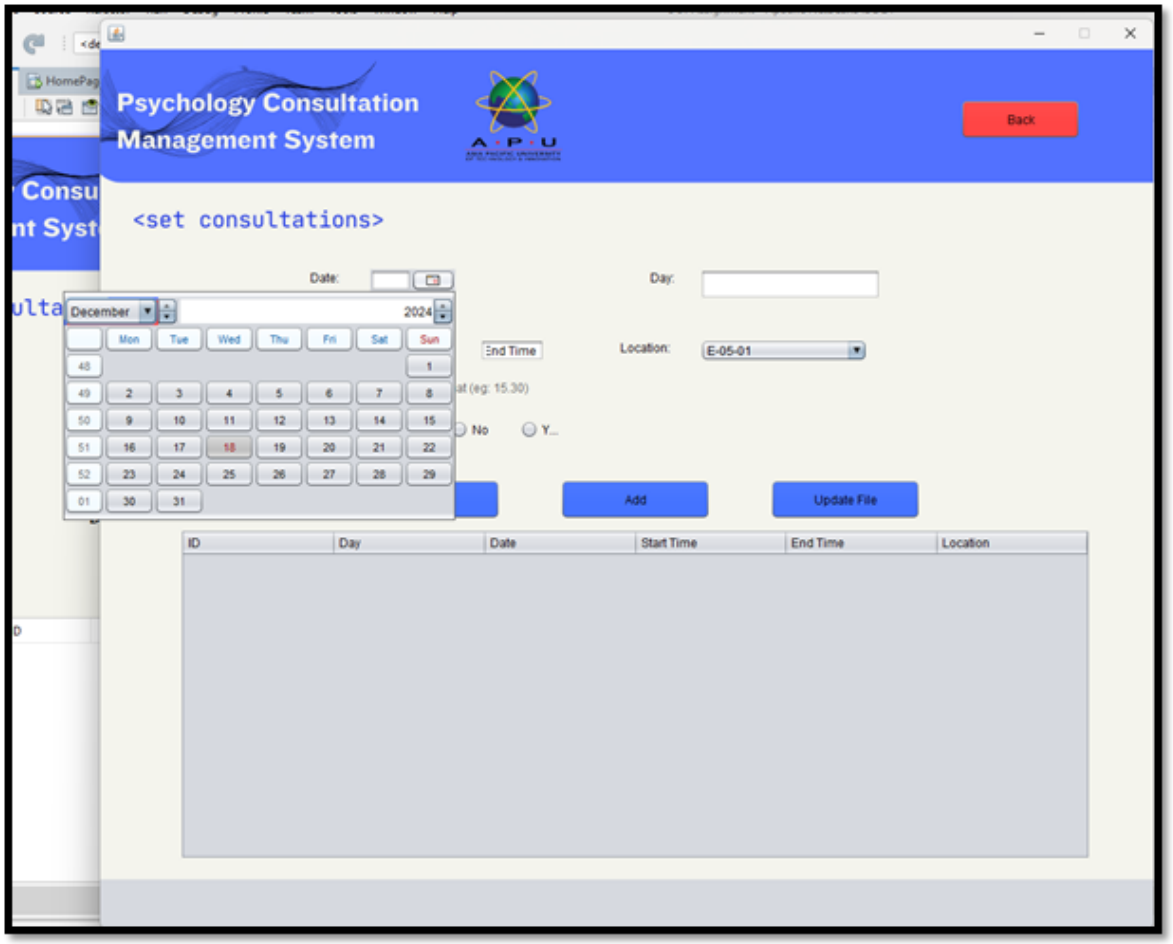
```java
public String getBookID(){
    return bookID;
}

public String getBookDay(){
    return bookDay;
}
public String getBookDate(){
    return bookDate;
}
public String getBookStartTime(){
    return bookStartTime;
}
public String getBookEndTime(){
    return bookEndTime;
}
public String getBookLocation(){
    return bookLocation;
}
public String getBookStatus(){
    return bookStatus;
}

public void setConsultDate(String newDate){
    this.bookDate = newDate;
}
public void setConsultStartTime(String newStartTime){
    this.bookStartTime = newStartTime;
}
public void setConsultEndTime(String newEndTime){
    this.bookEndTime = newEndTime;
}
public void setBookStatus(String newStatus){
    this.bookStatus = newStatus;
}
public String toString(){
        return bookID + ":" + bookDay + ":" + bookDate + ":" + bookStartTime + ":" +
                bookEndTime + ":" + bookLocation + ":" + bookStatus;
    }
}
```

Feedback Class

```java
public class Feedback extends Booking {
    private String feedbackID;
    private String feedbackLecturer;
    private String feedbackStudent;

public Feedback (String feedbackID, String bookDay, String bookDate, String bookStartTime, String bookEndTime, String bookLocation, String bookStatus,
        String feedbackLecturer, String feedbackStudent) {

super("bookID", bookDay, bookDate, bookStartTime, bookEndTime, bookLocation, bookStatus);

this.feedbackID = feedbackID;
this.feedbackLecturer = feedbackLecturer;
this.feedbackStudent = feedbackStudent;
}

public String getFeedbackID(){
    return feedbackID;
}

public String getFeedbackLecturer(){
    return feedbackLecturer;
}

public String getFeedbackStudent(){
    return feedbackStudent;
}

public void setFeedbackStudent(String completeStudentFeedback){
    this.feedbackStudent = completeStudentFeedback;
}

    @Override
    public String toString(){
    return feedbackID + ":" + getBookDay() + ":" + getBookDate() + ":" + getBookStartTime()
            + ":" + getBookEndTime() + ":" + getBookLocation() + ":" + getBookStatus() +
            ":" + feedbackLecturer + ":" + feedbackStudent;
}
}
```

Feedback Class inherits the attributes bookDay, bookDate, bookStartTime, bookEndTime, bookLocation and bookStatus from Booking Class alongside with their accompanying getters and setter methods.

5. Incorporation of Additional Features

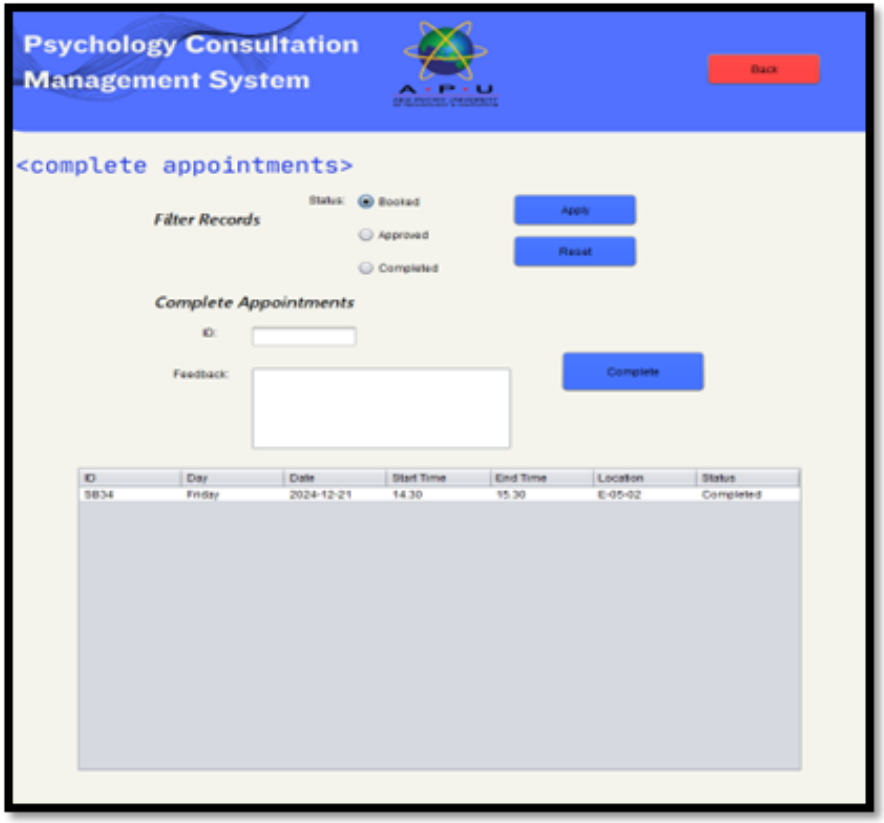
JCalender


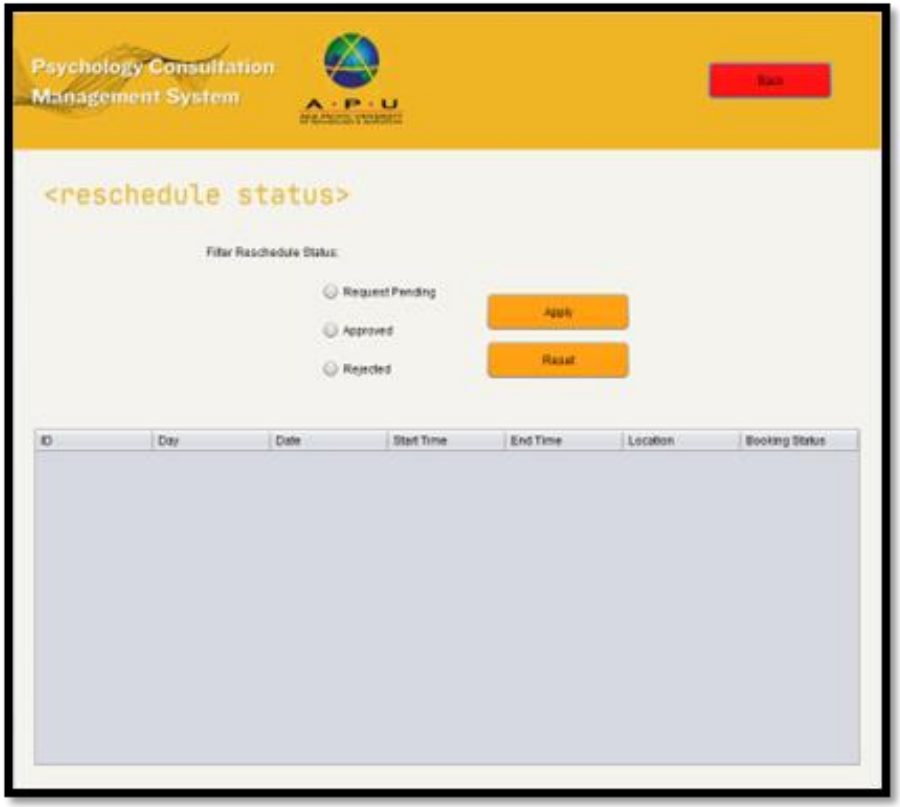
JCalendar has been implemented into the consultation management system in which students can select their consultation slots easily with the help of the calendar while teachers can utilize the JCalendar to add their consultation slots for students to view. The installation of JCalendar has made the system's interface more user friendly as users can directly select the dates instead of manually keying the dates in a text field. Through a JCalendar, it can elevate user experience during the booking process as it eases navigation usage. With the help of a visually appealing calendar, users can see the dates clearly, preventing them from entering the incorrect or invalid dates.

```
private void tfDatePropertyChange(java.beans.PropertyChangeEvent evt) {

    Date date = tfDate.getDate();

    if( date != null){
    SimpleDateFormat day = new SimpleDateFormat("EEEE");
    String consultDay = day.format(date);

    tfDay.setText(consultDay);

    }


}
```

After adding the JDateChooser into the Graphical User Interface (GUI), the code seen above is added to check the value entered by users. After that, it will display the corresponding days based on the dates that have been selected by user and sets the day text field. Users will not be able to edit the day text field so that a valid day will match the selected date.

FILTER RECORDS





Based on the figures above, we have added a filter function on both lecturer and student page for customers to optimize the interface. The filter function can help users to locate the desired information more efficiently and helps to organize the information in a systematic manner.

```java
private void bApplyFilterActionPerformed(java.awt.event.ActionEvent evt) {
    BookingFileHandler file = new BookingFileHandler();
    DefaultTableModel model = (DefaultTableModel) tAppointment.getModel();
    String Filter = null;

    if(rbBooked.isSelected()){
        Filter = "Booked";

    } else if (rbCompleted.isSelected()){
        Filter = "Completed";

    } else if (rbApproved.isSelected()) {
        Filter = "Approved";

    }



    try {
        List<Booking> existingSlots = file.fetchBookingData();
        model.setRowCount(0);
        for(Booking existingSlot : existingSlots){
        if (existingSlot.getBookStatus().equals(Filter)){
            model.addRow(new Object[]{
                existingSlot.getBookID(),
                existingSlot.getBookDay(),
                existingSlot.getBookDate(),
                existingSlot.getBookStartTime(),
                existingSlot.getBookEndTime(),
                existingSlot.getBookLocation(),
                existingSlot.getBookStatus()
            });
        }
        }
    } catch (IOException ex) {
        Logger.getLogger(CompleteAppointments.class.getName()).log(Level.SEVERE, null, ex);
    }

}
```

The filter displayed to users is in the form of radio buttons where the code is implemented after grouping every radio button in a group. A new instance of the BookingFileHandler is created and assigned to the variable ‘file’. Filter variable have been declared as a String to store the filter value for the booking status such as “Booked”, “Completed”, and “Approved”. Based on the filter value that users input from the radio button selection, the table at the Complete Appointment page will fetch the data from the booking.txt text file with the help of the fetchBookingData().

It will return all the booking details that have been stored and ready for users to view. The booking details fetched will be stored in the respective columns of the table.

```
private void btnApplyActionPerformed(java.awt.event.ActionEvent evt) {
    BookingFileHandler file = new BookingFileHandler();
    DefaultTableModel model = (DefaultTableModel) tAppoinment.getModel();
    String Filter = null;

    if(jPending.isSelected()){
        Filter = "Request Pending";

    } else if (jApproved.isSelected()){
        Filter = "Approved";

    } else if (jRejected.isSelected()){
        Filter = "Rejected";
    }

    try {
        List<Booking> existingSlots = file.fetchBookingData();
        model.setRowCount(0);
        for(Booking existingSlot : existingSlots){
        if (existingSlot.getBookStatus().equals(Filter)){
            model.addRow(new Object[]{
                existingSlot.getBookID(),
                existingSlot.getBookDay(),
                existingSlot.getBookDate(),
                existingSlot.getBookStartTime(),
                existingSlot.getBookEndTime(),
                existingSlot.getBookLocation(),
                existingSlot.getBookStatus()
            });
        }
        }
    } catch (IOException ex) {
        Logger.getLogger(CompleteAppointments.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Based on the figure above, the same source code for Complete Appointment filter is also used in Reschedule Status filter with slight changes to the name of the radio button.

6. Conclusion

The object-oriented programming principles have been successfully demonstrated in the psychology consultation management system. The system addresses the need for an online management system between students and lecturers by providing the basic functionalities such as user registration and appointment management. Moreover, this project has given our team the opportunity to implement key OOP concepts such as classes, objects, encapsulation and polymorphism. Through this experience, our team have gotten a better grasp on the concepts to be further applied in the near future as it has enhanced our understanding of an object-oriented approached and reinforced the importance of code readability and maintainability. By incorporating these features into the system, it would not only meet the current functional requirements but also allows future growth to adapt to the needs of users or feature expansions. The supporting documentation displayed has provided an overview of the implementation, complete with scenarios and code samples that highlight the use of OOP principles throughout the system. A systematic approach can be seen in designing, implementing and testing the program accordingly. Overall, our team managed to successfully fulfil the objectives of the project for the needs of students and lecturers. Furthermore, the project demonstrates a solid foundation in object-oriented programming and the ability to design an effective software solution with Java implementation.

Reference

Gaith Ibrahim Alshammare, M. S. (2022). Online Booking Services Assisted By Technology To Improve Customer Loyalty in Jordanian Five-Star Hotels. *International Journal of Professional Business Review*, 1-15.

Workload Matrix

Student 1:
Elianna Catrina Herrera (TP073631)
	- Developed the Student Side of the Program
	- Documentation:
	  o Introduction
	  o Sample Input Output (Student)
	  o OOP Concepts (Student)
	  o Extra Features (Student)
	  o Conclusion
	  o Documentation Compilation

Student 2:
Ejjaz Hakimi bin Mohamad Azan (TP073318)
	- Developed the Lecturer Side of the Program
	- Designed the UI for both Lecturer & Student
	- Compiled & debugged the code
	- Documentation:
	  o Sample Input & Output (Lecturer)
	  o OOP Concepts (Lecturer)
	  o Extra Features (Lecturer)