MULTIMEDIA UNIVERSITY

PROJECT

TOP2121 – OBJECT-ORIENTED PROGRAMMING

TRIMESTER 1, 2022/2023 (TERM 2210)

University Venue Management System

22 January 2023

NO	STUDENT ID	STUDENT NAME	MAJOR
1.	1191103341	Violette Lai Zi Yunn	ST
2.	1191103300	Evon Ng Yee Ting	ST
3.	1191203271	Ng Jin Yang	ST
4.	1191103098	Lee Chee Ann	ST

Table of Contents

Task Distribution 2
Introduction 3
Problem Statements 4
Objectives 4
Program Scope 5
  Features & Functions 5
  Deliverables 7
  Constraints 7
Detailed Description 8
  Detailed Explanation 8
  System Design 13
    Use-Case Diagram 13
    Class Diagram 14
  Draft Interface Design 15
Screenshot of the program 21
Reference 45
Appendix 46
  Source Code 46

Task Distribution

No.	Student ID	Student Name	Tasks Completed
1.	1191103341	Violette Lai Zi Yunn	• Task Distribution • System Design • Backend coding • Frontend coding
2.	1191103300	Evon Ng Yee Ting	• Detailed explanation • Draft Interface • Coding
3.	1191203271	Ng Jin Yang	• Program Features • Program Functions • Program Deliverables • Program Constraints
4.	1191103098	Lee Chee Ann	• Introduction • Problem statement • Objectives • UML Diagrams

Introduction

The project titled University Venue Management System enables an alternative way that allows university students to communicate with the university's administrator about booking venues on the campus. To prevent data loss after the program terminated, they are stored in a specific text file. There are two main actors in this program which are Student and Admin.

Students are required to login with required details such as username and password in order to reserve an available venue. They are able to edit, view and cancel their reserved requests on the student dashboard. However, reservation requests are only considered valid only after they are approved by an admin.

Admin is required to login with a specific password before entering Admin's menu interface. An admin is capable of processing requests, managing reservation records, and managing venue information. Admin can approve or deny pending requests, manually add a student's venue reservation request on the spot with instant approval, edit any existing records, view available system records, and cancel reservation records. Admin can also add new venues, edit venue information, view all venues, and delete a venue record.

Problem Statements

1. Students are unable to book a venue because they are busy with their class schedule in the standard working hour and not able to meet the administrator as they are away from the institution after working hours.
2. Reservation through pen and paper method is highly inefficient due to high probability of time clashes between timeslots.
3. Reservation details are easily forgotten by the students and the administrator.

Objectives

1. Students are able to reserve venues with the proposed system and the administrator can approve the request anytime.
2. The program reduces the probability of time clashes between reservation timeslots by restricting input.
3. The program is able to view all the venues booked with booking ID, venue ID, date and time booked, description of booking the venue, status of the bookings, request date and requestor's ID.

Program Scope

Features & Functions

- Add Function
    - Students may add a pending reservation request into the system to book an available timeslot, then wait for admin approval.
    - Admin can add new records manually to the system. This is to simulate a real-life situation where a staff adds a student's venue request on the spot with instant approval.
    - Admin may also add new venue information into the system for the students to reserve.
    - In both circumstances, only venue added into the system can be reserved.

- Edit Function
    - Students may edit their request to overwrite previous booking time slot and reservation details.
    - Admin can edit and update any existing reservation records in the system. Changes made by admin is absolute and approved instantly.
    - Admin can edit a venue record to update its information in the system, in case if any wrong venue information is found.

- View Function
    - Students may view details of all their reservation requests in student dashboard.
    - Admin may view details of all existing reservation records and venue bookings in the admin dashboard.
    - Admin may also view all available venue records and their information in the admin dashboard.

- Delete Function
    - Students are only allowed to cancel a reservation request they previously made.
    - Admin is also not allowed to completely remove a reservation record from the system. Only cancellation of a reservation is allowed.

  - Admin may remove a venue record completely from the system, in case of maintenance or permanent closure, so that users cannot reserve the removed venue anymore.

- User-friendly Graphical User Interface (GUI)
  - The GUI of the program is integrated to improve user friendliness.
  - The main menu and view function is merged together into a dashboard.
  - Users may click on any records displayed on the dashboard to perform further action, such as editing and deleting.

- Time Validation Feature
  - During the process of adding and editing reservation records, the system validates input date and time by checking its format (dd-mm-yyyy hh:mm)
  - The system also checks if the end date and time is the same with or lesser than the start date and time.
  - The system will only proceed to perform its operation (adding record or editing record) if the date and time is valid.

- Check Time Clash Feature
  - During the process of adding and editing reservation records, the check time clash feature is implemented.
  - This feature checks if the newly-input date and time clashes with any approved booking timeslots.
  - Users cannot proceed to add or save changes as long as time clash between timeslots occur.

- View Venue Timetable Feature
  - During the process of adding and editing reservation records, view venue timetable feature is implemented.
  - There is a “Timetable” button beside the venue option list. Clicking the button displays the current reservation timetable of a venue.
  - User can easily check for availability of a timeslot.

Deliverables

The system is a platform for users in the institution to book venue on the campus easily. Using this platform, students can check the venue timetable for their convenience to book slots for any purpose. Venue availability depends on the venue record within the system. After submitting a booking request, they can view, edit, and cancel their request. If anything happens, they can make modifications to their bookings. However, they are required to wait until their request is further approved or denied; else, the reservation is still considered as invalid.

For the admin side, they can edit and modify any booking made by. Admins are able to approve or deny any booking request. They could also add a reservation record on the spot. Furthermore, the admin also can add new additional venues to the system or modify any of the venue. The admin also can delete any venues from the list from the admin side.

Constraints

- **Reset Password Feature**

  The system lacks a password reset feature; hence the users are not able to edit or reset their password if they forget how to login. The only solution is to create a new account.

- **Edit User Information Feature**

  The system also lacks the edit user's information feature; hence the users are not able to edit their personal information such as username or password. The only solution now is asking them to register for a new account.

Detailed Description

Detailed Explanation

1. **Login**

There are 2 types of user account, student and admin. User must choose one of the login methods in order to use the system. For student account, the user needs to create their own account if they do not have one. They can enter anything for the username except for admin and existing username; any password is accepted. For admin account, it has only one account with username admin and the default password is admin. Creation of new admin account is not allowed.

2. **Menu and Dashboard**

The main menu is the login menu. After user login, the submenu or user's dashboard will be display depends on the login account type. For student account, dashboard will be displayed after login. It contains a list of all requests made by themselves that display in table form. On top of the table is the New Reservation Request and Log Out buttons. To edit a request, users can just click on the row of a record they want to edit. For admin account, it has one submenu called Admin Main Menu. The actions available in this menu are Manage Pending Request, Manage Records, Manage Venue and Log Out. Admin has the rights to process students' requests and also manage the venues available for reserve. Dashboard will also be displayed for Manage Pending Request, Manage Records and Manage Venue. The actions are similar as Student Dashboard except for Manage Pending Request. Manage Pending Request shows all requests that is under pending status and admin has to click on a row to approve or deny the request. This is to give admin an easier and quicker way to approve the requests. Users can click a column header to sort the data by that column.

3. **Add**

The record ID will be auto-generated when adding a reservation. Users are able to select any existed venue ID and click on Timetable button to view all approved reservations made to that venue, so that user can know which time slots the venue is not available for booking. Users need to enter the start and end date and time they want to reserve. It is optional for user to fill in a brief description of that reservation and "None" will be inserted if user does not enter any value. Status will be set as Pending by default as it is a new record that not yet approved by the admin. The request date and user ID of the requester will be recorded.

For adding a venue, user requires to enter a unique venue ID. After that, they can fill in the venue name, venue type, which are Tutorial Room, Lecture Room, Lecture Hall and Laboratory, and description of the venue. Click Cancel button will not add the new record and return to previous page.

4. Edit

To edit a record, users need to click on a row of a record on the dashboard table. After that, program will open the edit page and display all information of that record. Only some information is editable. For example, in edit reservation record, the editable fields are venue ID, start date and time, end date and time, and description. Record ID is auto-generated and is not allow to change. Status of a reservation will only be changed based on action of user. Request Date will automatically be updated after user save the edit. User ID is also fixed. The Cancel Request button will change the status of request to Cancelled. Go Back will make the record unchanged and exit the edit page, then return back to previous dashboard page.

For venue record, the venue ID is unable to edit after added. All other fields are editable. If the Venue Name and Description have no value, they will automatically be inserted with the value None. The Delete button will remove the venue record. Click Cancel button will go back to Manage Venue dashboard.

5. Delete

Admin and student only can cancel a reservation request and cannot completely remove a reservation record. This is because if a request is deleted, student cannot know if their request is being removed and will cause confusion to them. Therefore, the delete function of reservation record is only change the status to Cancelled and the record is kept. While for delete function of venue, the venue record will be permanently removed from the record file and users are unable to make reservation to that venue anymore.

6. View

Viewing and displaying of records is in the form of table. The records are mostly display at dashboard pages for users to quickly have a view and check their reservation requests easily. The table rows are clickable so that users no need to remember the record ID when they want to make changes to the record but the program will open edit page of the record ID of the clicked row. The table is sortable by clicking on a column header and the table data will be sorted accordingly. By default, it is in ascending order and click the column header again will switch the order from ascending to descending and vice versa. The table is scrollable as the list may be too long and cannot fit in the window.

7. Check Time Clash

This feature is used in add and edit reservation record, as well as in Manage Pending Requests. Since many users are submitting booking requests simultaneously, it is important to make sure there is no clashing reservation time for each venue. When adding a new reservation record, a Timetable button at the right side of venue ID field is for users to check which time slots are not available for booking. Only request with Approved status will be counted for checking time clash. Hence, they need choose any other free time slots and the time must be larger than the current time as users are not allow to make a booking where the time is passed. The duration is not restricted, whereby they can enter any start and end date time as long as it is not clashing with existing records. The date and time format is date-month-year hour:minute (dd-MM-yyyy hh:mm). Program will check if user enter a valid date format. It will start check from start date time, followed by checking end date time, then lastly check overlap time clash, where the input date time cannot be covering existing reservation time. For example, the approved time slots is 22-01-2023 12:00 to 14:00, hence a new request reservation time of 22-01-2023 10:00 to 16:00 is considered overlap clashing. The function also will check if the end date time is before or smaller than its start date time which is unacceptable.

8. Request Status

There are 4 types of status for each request, which are Pending, Approved, Cancelled and Denied. A request is in Pending status when students submit a new request and when they edit a request. As save the edit is considered as resubmitting the reservation request, the status will change to Pending regardless of the original status the request is in. For example, if user edits an Approved request by changing the date and time, the status will be changed from Approved to Pending. Only admin can make a request become Approved. There are 2 ways admin can approve a request. First is go to Manage Pending Requests where admin can view all Pending requests and choose to approve the requests with no time clash. Second is at edit page under Manage Records where saving an edit of record will make the request to be Approved regardless of its original status. Both student and admin can cancel a request by clicking Cancel Request button in edit page to make a record in Cancelled status if users do not want the request to be processed for approving. A request will only be in Denied status when admin click No in confirmation of Manage Pending Requests which means the request is not being approved and rejected.

9. GUI

We are mainly using Java Swing to build the system interface. Swing provides more flexibility than Applet. The buttons and labels can be added to a panel and adjust their size and position within the panel. As we are using only one frame at a time, all panels will be added into the frame and the GUI components will be displayed nicely. Each button is handled with the Listener functions to perform specific actions when users clicked on it. JTextField is used to accept user input and JComboBox is to hold a list of data in a dropdown list for user to choose one of it. Null will be the default value if there is no data to hold by JComboBox. To prevent users from accidentally close the program, a pop out window will come out when users click on the top right close button of the window to confirm with the users if they really want to exit and close the program. Since the frame is kept on changing, the frame title will be changed when opening a new frame window to make user clear with where they are.

10. File Handling

There are some functions in the program to handle the file records, including countRecord, loadRecord, writeFile and updateRec. countRecord calculates the total number of records in a text file for determining the current ID number by adding 1 to the count result. loadRecord will get and load all the records in a text file and store them to a vector of string for later use in the program for ease of access and change the records. writeFile will updates the text file content when the records are change in the program. updateRec is to save changes of records.

11. Exception Handling

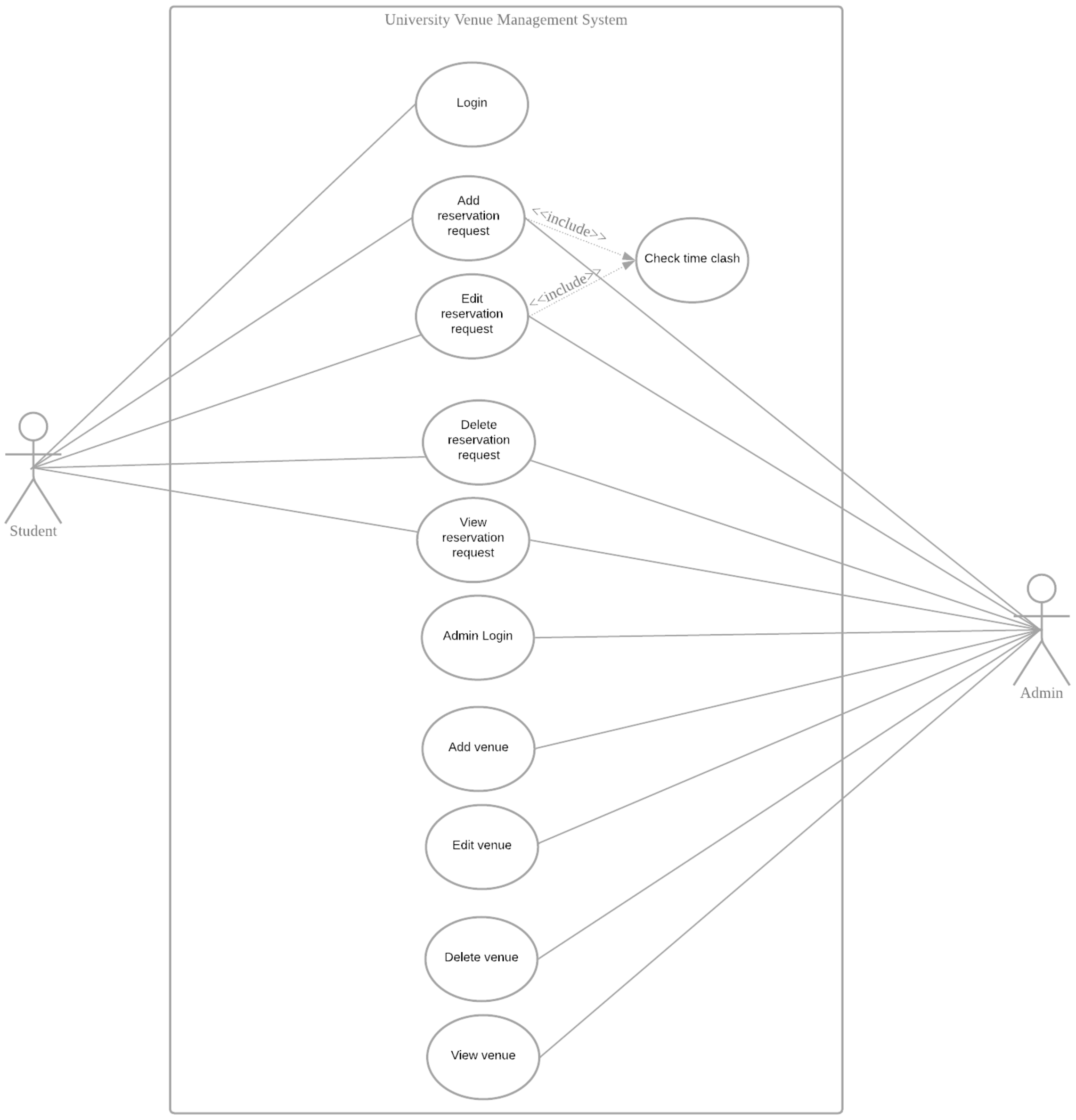
Exception handling is use when reading and writing a file. It is also used when accepting input, validating a date and converting a date. The possible errors are handled so that the program will not stop abruptly. Warning or error messages will be shown to user to give users some hints in using the program correctly.

12. Classes

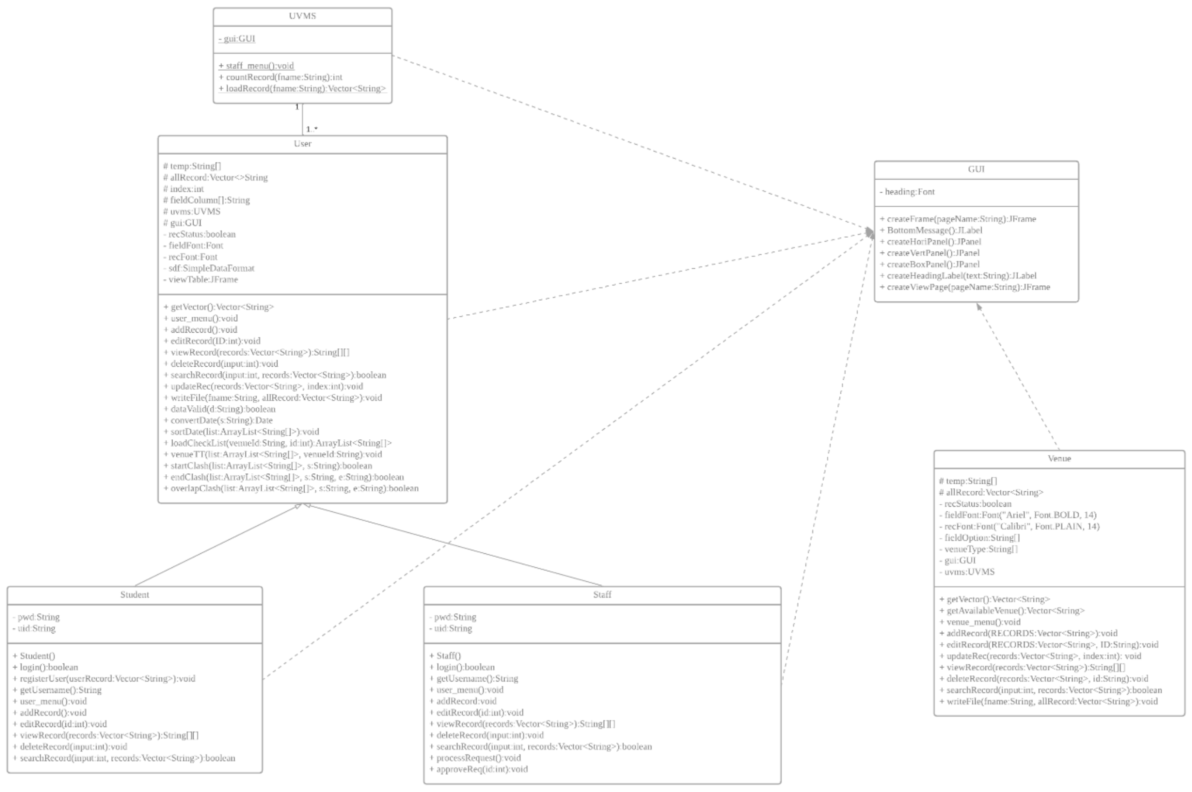
The User class is an abstract class. It provides functions to manage the reservation record which can be overridden by its subclass. Staff and Student are inherited from User. They have their own extra data and functions. For example, processRequest and approveReq are the extra functions own by Staff class because only Staff user can perform the Manage Pending Request action. When the child classes want to use or invoke parent class properties rather than its own, the keyword super is used. Venue class is a class used for instantiating and manipulating venue object. GUI class provides functions to create GUI components. UVMS class contains the main function and menus.

System Design

Use-Case Diagram

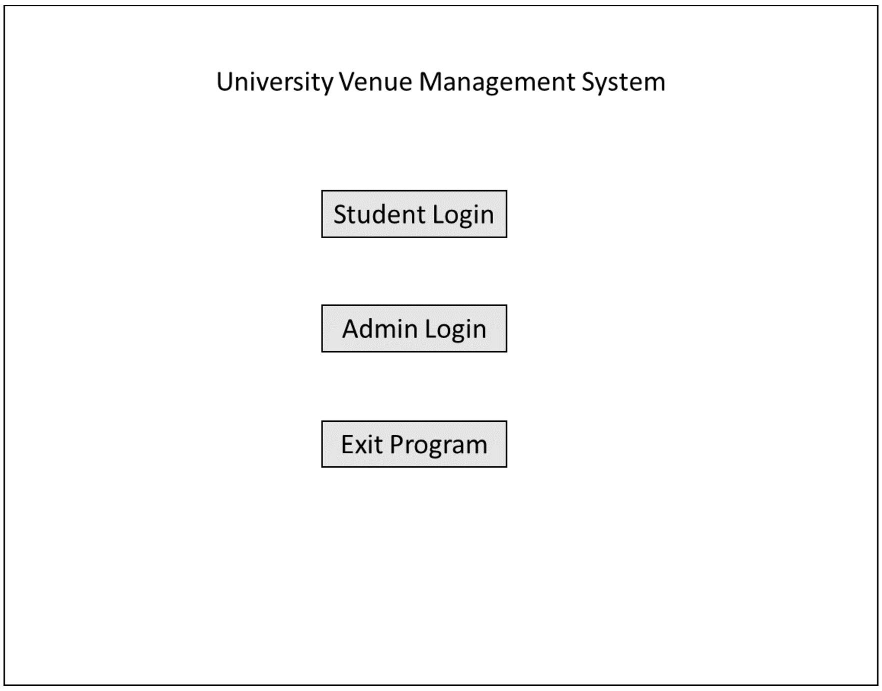


Class Diagram

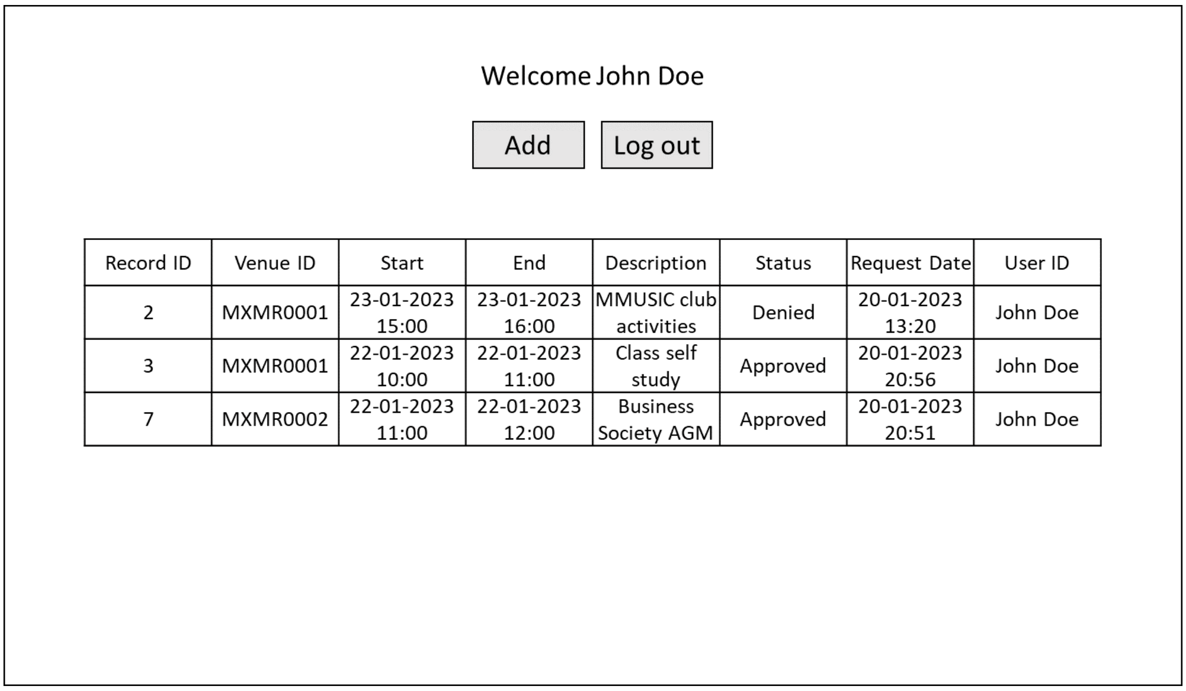

UVMS
- gui:GUI
+ staff_menu():void
+ countRecord(fname:String):int
+ loadRecord(fname:String):Vector<String>
1
1..*
User
# temp:String[]
# allRecord:Vector<>String
# index:int
# fieldColumn[]:String
# uvms:UVMS
# gui:GUI
- recStatus:boolean
- fieldFont:Font
- recFont:Font
- sdf:SimpleDataFormat
- viewTable:JFrame
+ getVector():Vector<String>
+ user_menu():void
+ addRecord():void
+ editRecord(ID:int):void
+ viewRecord(records:Vector<String>):String[][]
+ deleteRecord(input:int):void
+ searchRecord(input:int, records:Vector<String>):boolean
+ updateRec(records:Vector<String>, index:int):void
+ writeFile(fname:String, allRecord:Vector<String>):void
+ dataValid(d:String):boolean
+ convertDate(s:String):Date
+ sortDate(list:ArrayList<String[]>):void
+ loadCheckList(venueId:String, id:int):ArrayList<String[]>
+ venueTT(list:ArrayList<String[]>, venueId:String):void
+ startClash(list:ArrayList<String[]>, s:String):boolean
+ endClash(list:ArrayList<String[]>, s:String, e:String):boolean
+ overlapClash(list:ArrayList<String[]>, s:String, e:String):boolean
GUI
- heading:Font
+ createFrame(pageName:String):JFrame
+ BottomMessage():JLabel
+ createHoriPanel():JPanel
+ createVertPanel():JPanel
+ createBoxPanel():JPanel
+ createHeadingLabel(text:String):JLabel
+ createViewPage(pageName:String):JFrame
Venue
# temp:String[]
# allRecord:Vector<String>
- recStatus:boolean
- fieldFont:Font("Ariel", Font.BOLD, 14)
- recFont:Font("Calibri", Font.PLAIN, 14)
- fieldOption:String[]
- venueType:String[]
- gui:GUI
- uvms:UVMS
+ getVector():Vector<String>
+ getAvailableVenue():Vector<String>
+ venue_menu():void
+ addRecord(RECORDS:Vector<String>):void
+ editRecord(RECORDS:Vector<String>, ID:String):void
+ updateRec(records:Vector<String>, index:int): void
+ viewRecord(records:Vector<String>):String[][]
+ deleteRecord(records:Vector<String>, id:String):void
+ searchRecord(input:int, records:Vector<String>):boolean
+ writeFile(fname:String, allRecord:Vector<String>):void
Student
- pwd:String
- uid:String
+ Student()
+ login():boolean
+ registerUser(userRecord:Vector<String>):void
+ getUsername():String
+ user_menu():void
+ addRecord():void
+ editRecord(id:int):void
+ viewRecord(records:Vector<String>):String[][]
+ deleteRecord(input:int):void
+ searchRecord(input:int, records:Vector<String>):boolean
Staff
- pwd:String
- uid:String
+ Staff()
+ login():boolean
+ getUsername():String
+ user_menu():void
+ addRecord:void
+ editRecord(id:int):void
+ viewRecord(records:Vector<String>):String[][]
+ deleteRecord(input:int):void
+ searchRecord(input:int, records:Vector<String>):boolean
+ processRequest():void
+ approveReq(id:int):void

Draft Interface Design

1. Login Page

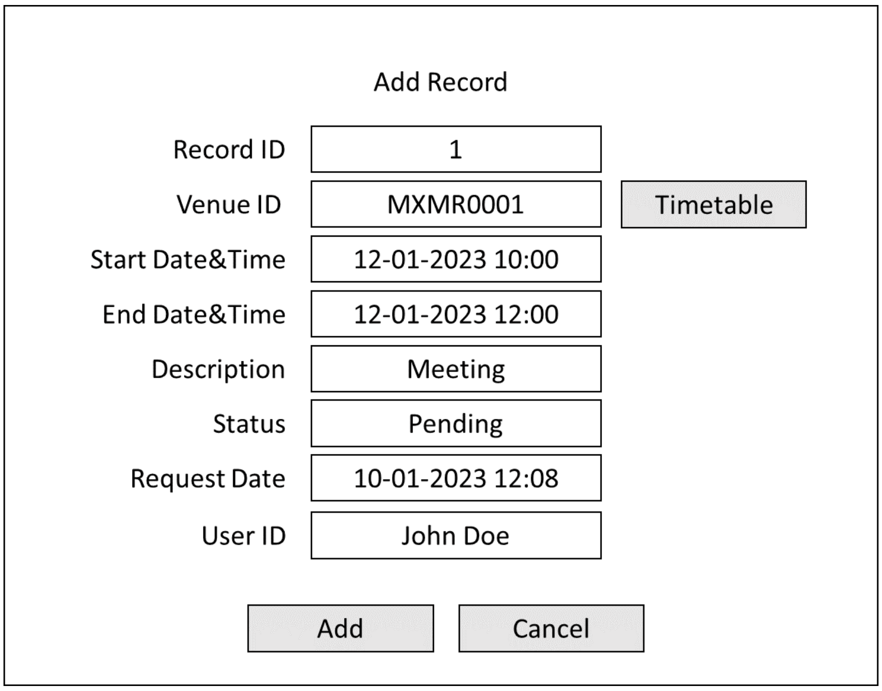

University Venue Management System

2. Student Main Menu


Welcome John Doe

Record ID	Venue ID	Start	End	Description	Status	Request Date	User ID
2	MXMR0001	23-01-2023 15:00	23-01-2023 16:00	MMUSIC club activities	Denied	20-01-2023 13:20	John Doe
3	MXMR0001	22-01-2023 10:00	22-01-2023 11:00	Class self study	Approved	20-01-2023 20:56	John Doe
7	MXMR0002	22-01-2023 11:00	22-01-2023 12:00	Business Society AGM	Approved	20-01-2023 20:51	John Doe

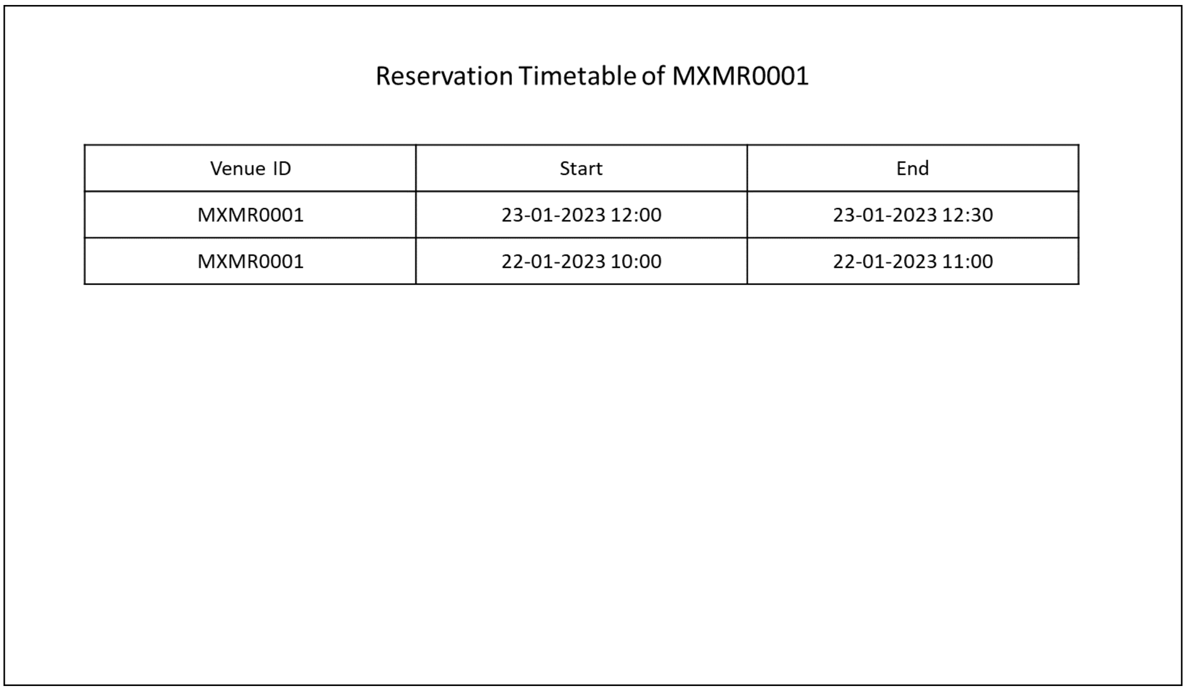
3. Add Record



Add Record

Record ID	1	
Venue ID	MXMR0001	Timetable
Start Date&Time	12-01-2023 10:00	
End Date&Time	12-01-2023 12:00	
Description	Meeting	
Status	Pending	
Request Date	10-01-2023 12:08	
User ID	John Doe	

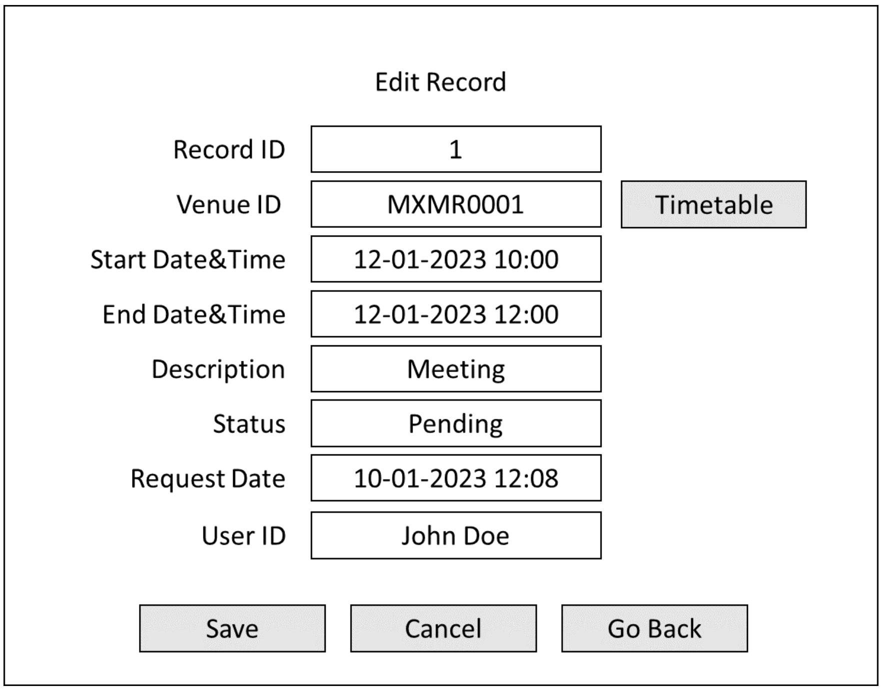
4. Reservation Timetable



Reservation Timetable of MXMR0001

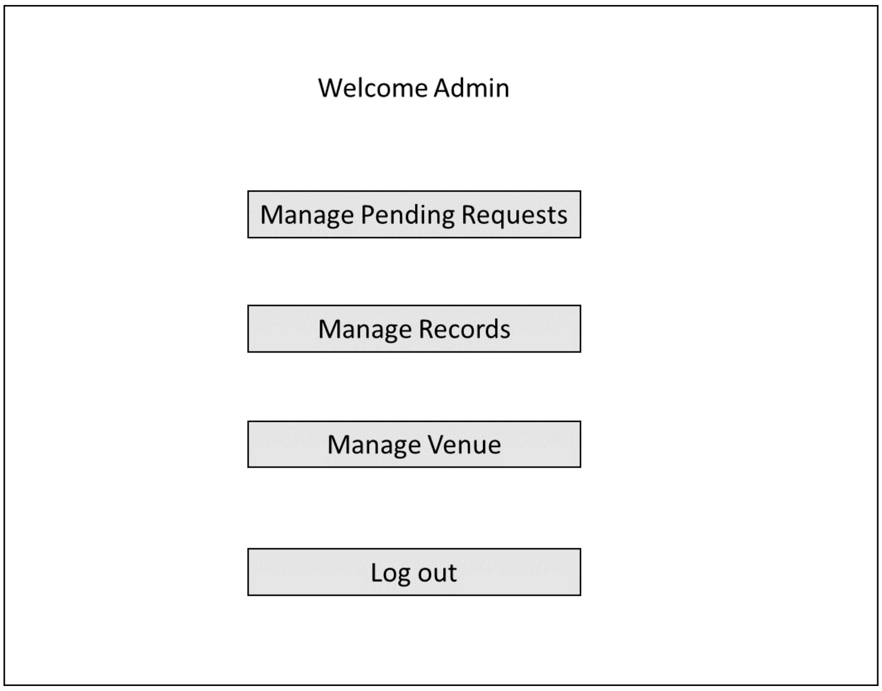
Venue ID	Start	End
MXMR0001	23-01-2023 12:00	23-01-2023 12:30
MXMR0001	22-01-2023 10:00	22-01-2023 11:00

5. Edit Record

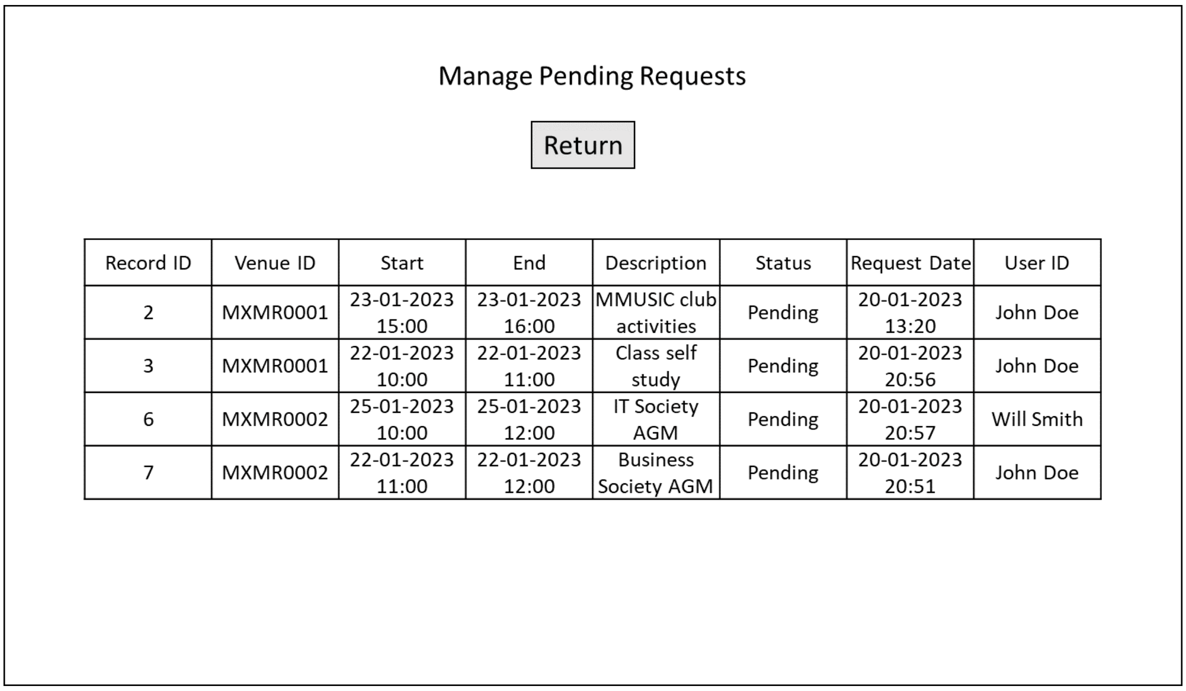

Edit Record

Record ID	1
Venue ID	MXMR0001
Start Date&Time	12-01-2023 10:00
End Date&Time	12-01-2023 12:00
Description	Meeting
Status	Pending
Request Date	10-01-2023 12:08
User ID	John Doe

6. Admin Menu


Welcome Admin

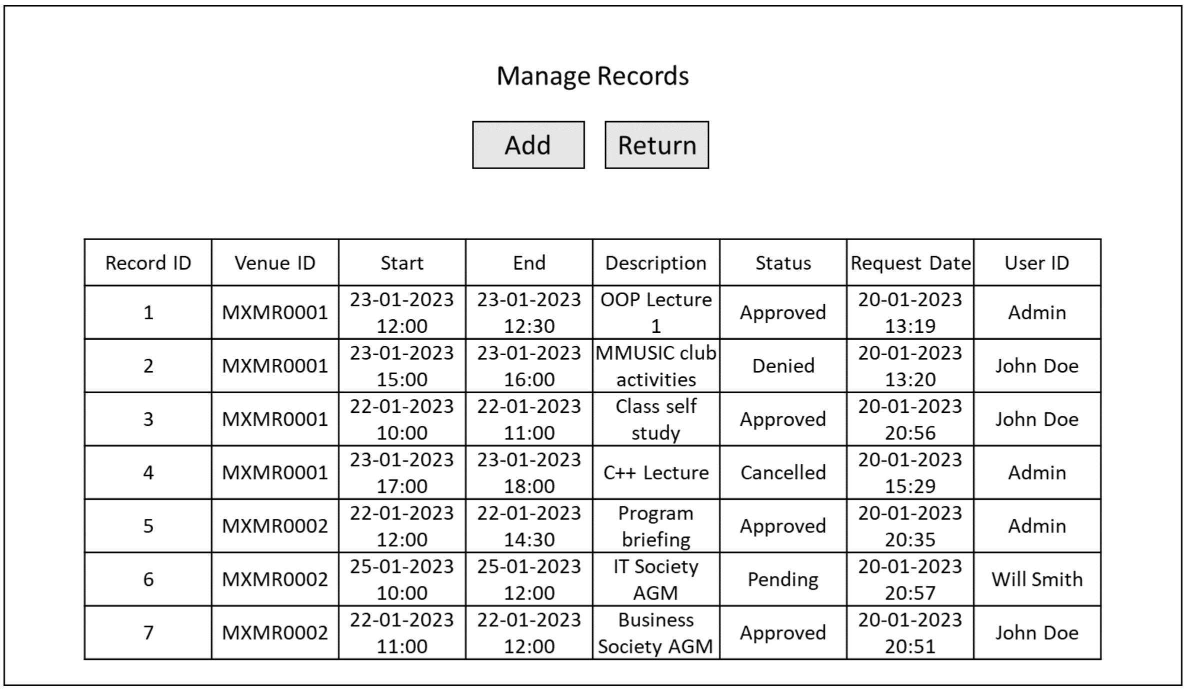
7. Manage Pending Requests Menu



Manage Pending Requests

Record ID	Venue ID	Start	End	Description	Status	Request Date	User ID
2	MXMR0001	23-01-2023 15:00	23-01-2023 16:00	MMUSIC club activities	Pending	20-01-2023 13:20	John Doe
3	MXMR0001	22-01-2023 10:00	22-01-2023 11:00	Class self study	Pending	20-01-2023 20:56	John Doe
6	MXMR0002	25-01-2023 10:00	25-01-2023 12:00	IT Society AGM	Pending	20-01-2023 20:57	Will Smith
7	MXMR0002	22-01-2023 11:00	22-01-2023 12:00	Business Society AGM	Pending	20-01-2023 20:51	John Doe

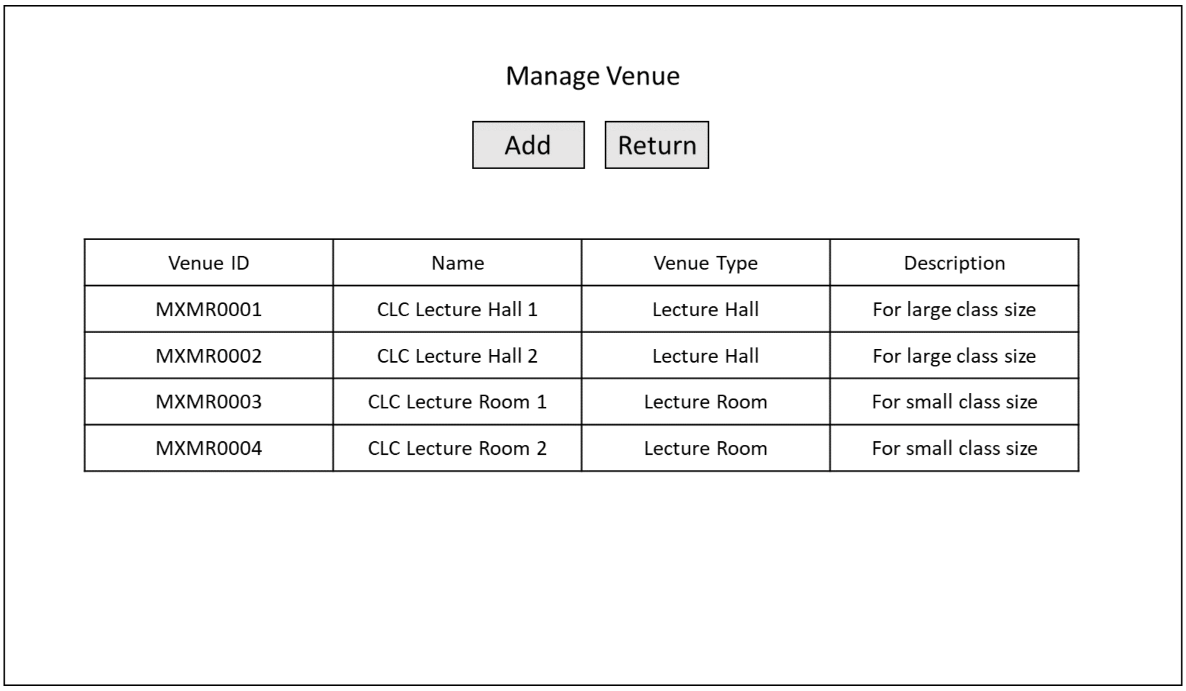
8. Manage Records Menu



Manage Records

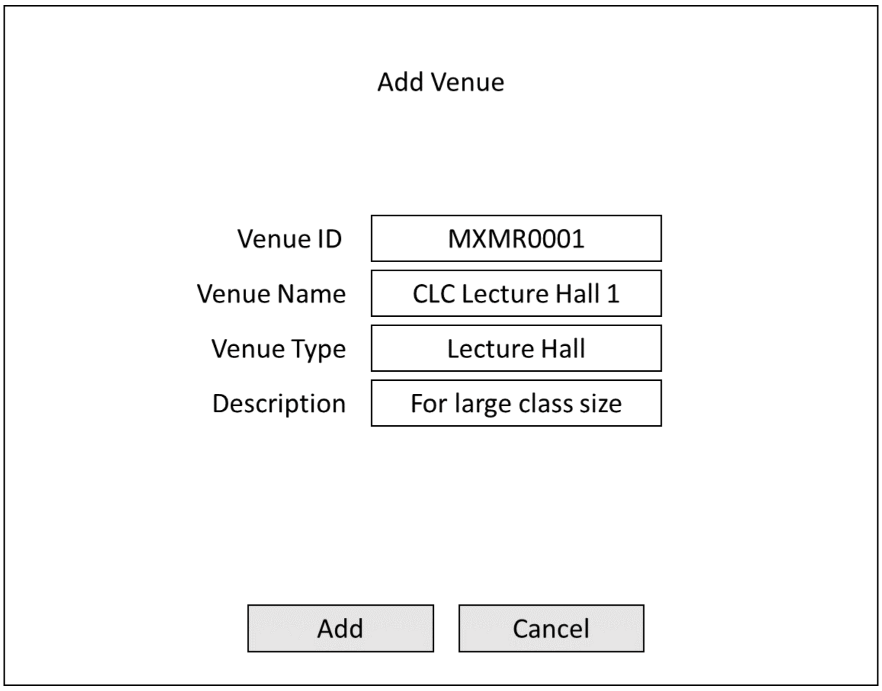
Record ID	Venue ID	Start	End	Description	Status	Request Date	User ID
1	MXMR0001	23-01-2023 12:00	23-01-2023 12:30	OOP Lecture 1	Approved	20-01-2023 13:19	Admin
2	MXMR0001	23-01-2023 15:00	23-01-2023 16:00	MMUSIC club activities	Denied	20-01-2023 13:20	John Doe
3	MXMR0001	22-01-2023 10:00	22-01-2023 11:00	Class self study	Approved	20-01-2023 20:56	John Doe
4	MXMR0001	23-01-2023 17:00	23-01-2023 18:00	C++ Lecture	Cancelled	20-01-2023 15:29	Admin
5	MXMR0002	22-01-2023 12:00	22-01-2023 14:30	Program briefing	Approved	20-01-2023 20:35	Admin
6	MXMR0002	25-01-2023 10:00	25-01-2023 12:00	IT Society AGM	Pending	20-01-2023 20:57	Will Smith
7	MXMR0002	22-01-2023 11:00	22-01-2023 12:00	Business Society AGM	Approved	20-01-2023 20:51	John Doe

9. Manage Venue Menu



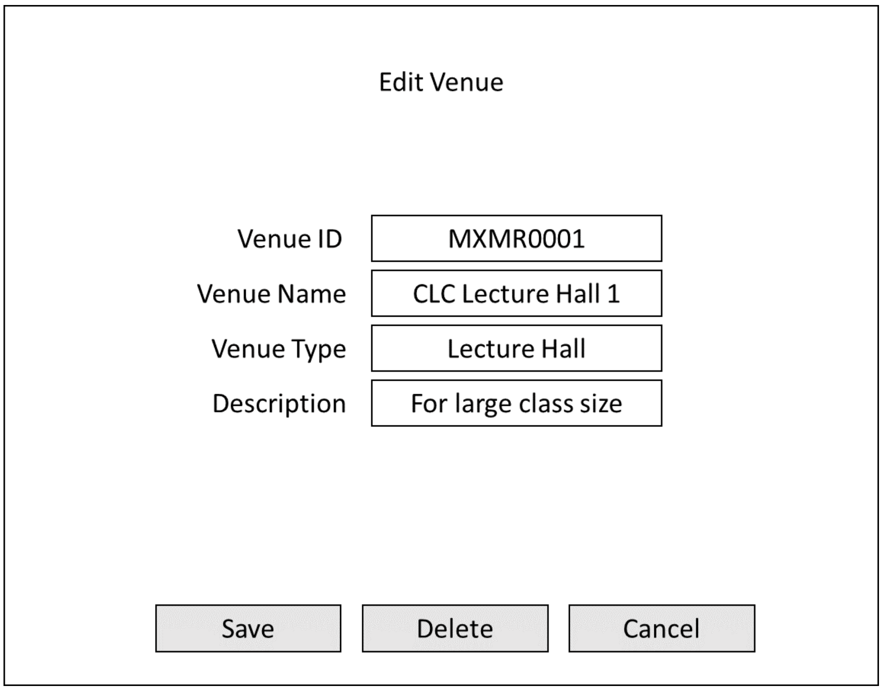
Venue ID	Name	Venue Type	Description
MXMR0001	CLC Lecture Hall 1	Lecture Hall	For large class size
MXMR0002	CLC Lecture Hall 2	Lecture Hall	For large class size
MXMR0003	CLC Lecture Room 1	Lecture Room	For small class size
MXMR0004	CLC Lecture Room 2	Lecture Room	For small class size

10. Add Venue



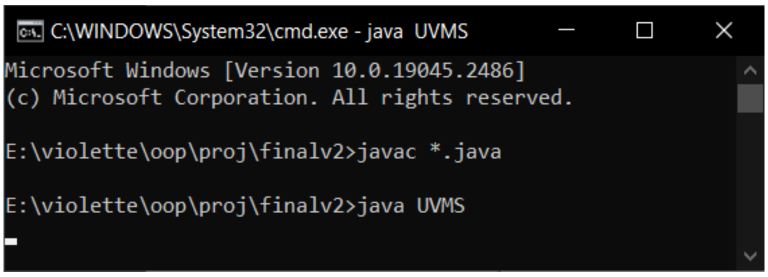
Venue ID	MXMR0001
Venue Name	CLC Lecture Hall 1
Venue Type	Lecture Hall
Description	For large class size

11. Edit Venue


Edit Venue

Venue ID	MXMR0001
Venue Name	CLC Lecture Hall 1
Venue Type	Lecture Hall
Description	For large class size

Screenshot of the program



```
C:\WINDOWS\System32\cmd.exe - java  UVMS
Microsoft Windows [Version 10.0.19045.2486]
(c) Microsoft Corporation. All rights reserved.

E:\violette\oop\proj\finalv2>javac *.java

E:\violette\oop\proj\finalv2>java UVMS
```

Figure 1.0 Launch Program

To start the GUI, compile all .java files and run UVMS.class

1.0 Login Page

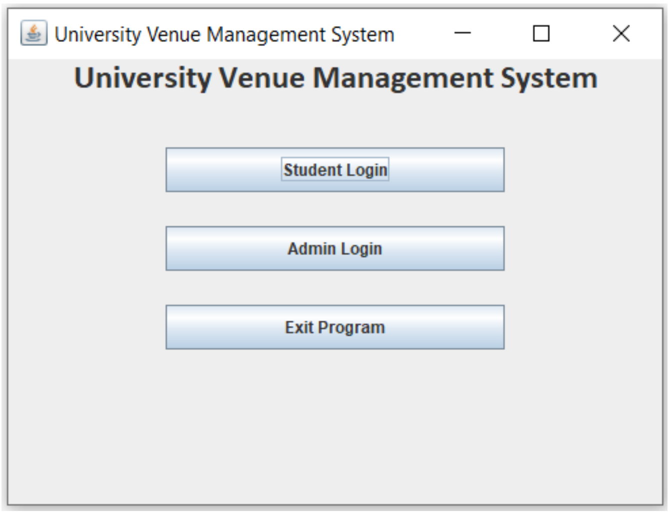
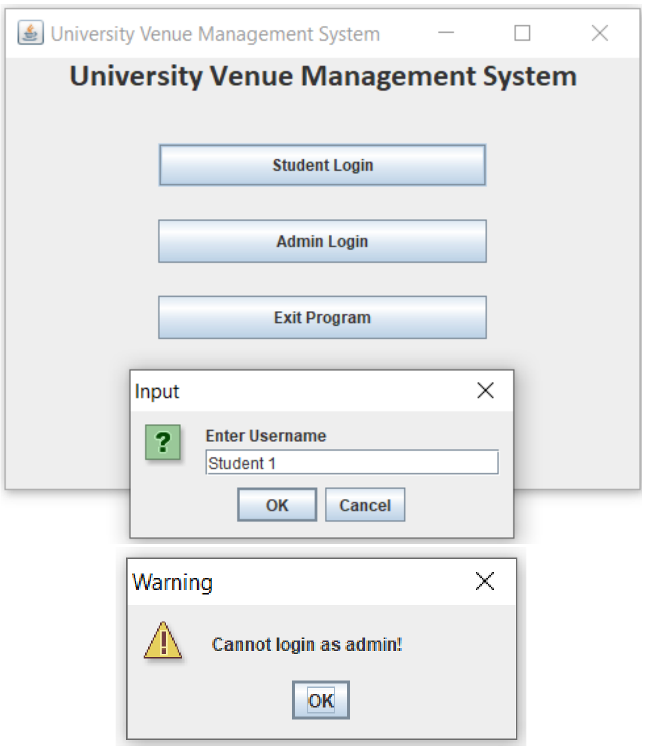

University Venue Management System
University Venue Management System
Student Login
Admin Login
Exit Program

Figure 1.0.1 Login Page

Login buttons prompt student and admin login respectively. Exit button terminates the program.

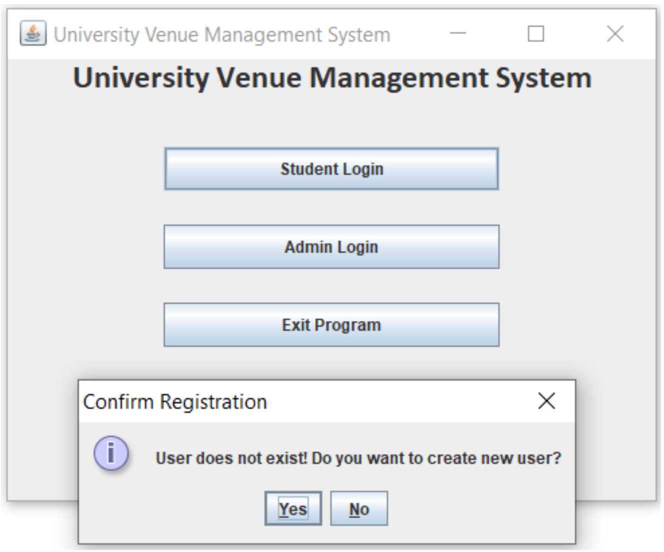
1.1 Student Login



Cannot login as admin!

Figure 1.1.1 Student Login

User is required to enter a username to login. User is not allowed to login as admin. The login process is case sensitive.


University Venue Management System
University Venue Management System
Student Login
Admin Login
Exit Program
Confirm Registration
Yes
No

User does not exist! Do you want to create new user?

Figure 1.1.2 Student Login

If the username is not a registered user, user may choose to register a new user.

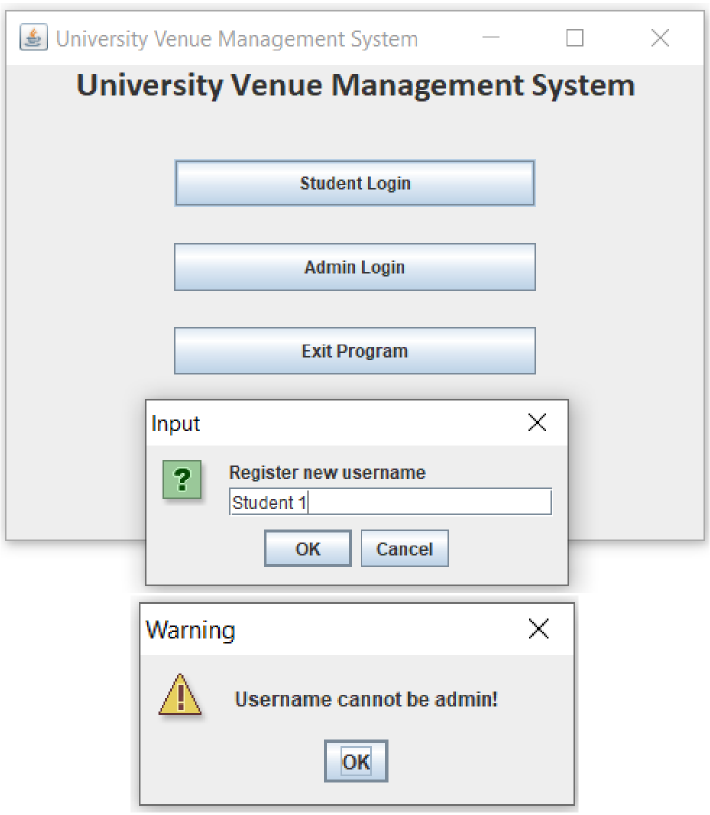
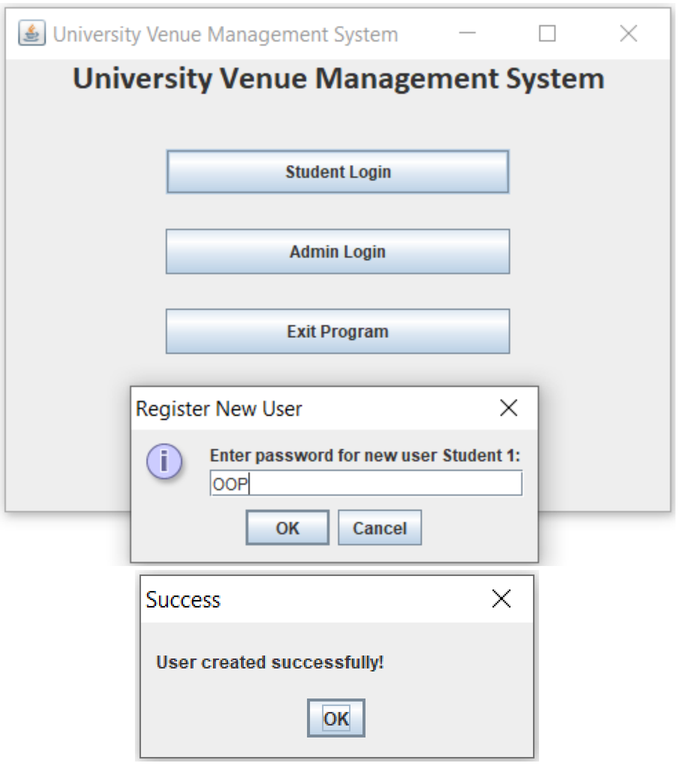

University Venue Management System
University Venue Management System
Student Login
Admin Login
Exit Program
Input
Register new username
Student 1
OK
Cancel
Warning
Username cannot be admin!
OK

Figure 1.1.3 Student Registration

User is asked for a new username. Username cannot be admin or is an existing username. Username and password are case sensitive.



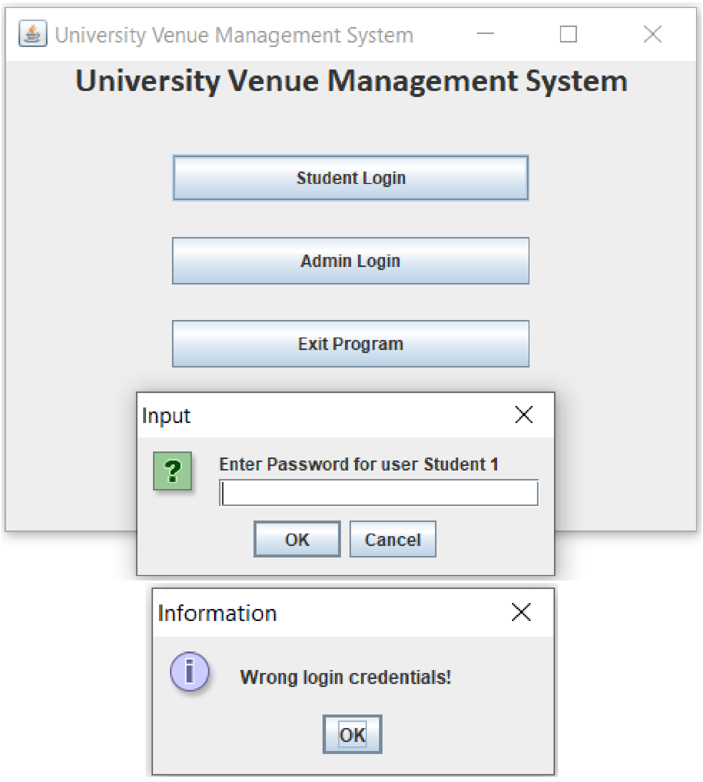
Enter password for new user Student 1:

OOP

User created successfully!

Figure 1.1.4 Student Registration

User is asked to input password for new user. New user is created after entering password.


University Venue Management System
University Venue Management System
Student Login
Admin Login
Exit Program
Input
OK
Cancel
Information
OK

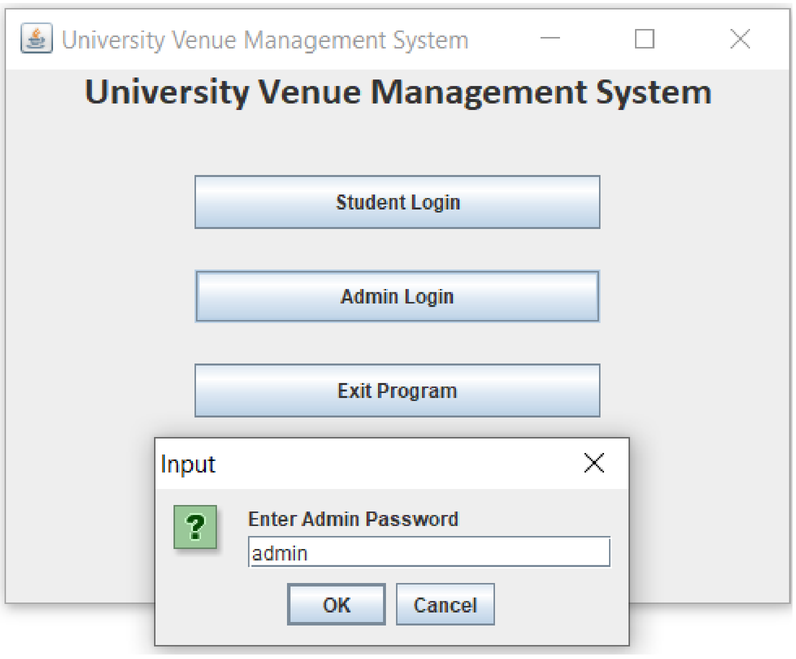
Enter Password for user Student 1

Wrong login credentials!

Figure 1.1.5 Student Login

User is asked to enter password for the input username. Program checks if login credentials match.

1.2 Admin Login


University Venue Management System
University Venue Management System
Student Login
Admin Login
Exit Program
Input
OK
Cancel

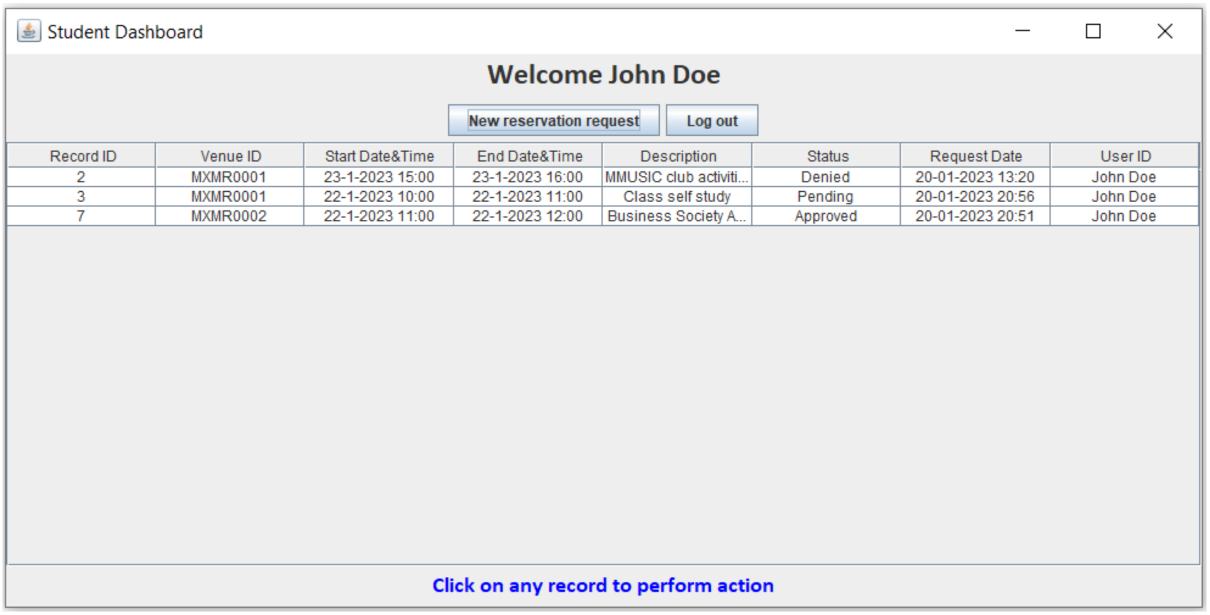
Enter Admin Password

admin

Figure 1.2.1 Admin Login

User is asked for admin password. Default password for admin is “admin”.

2.0 Student



Welcome John Doe

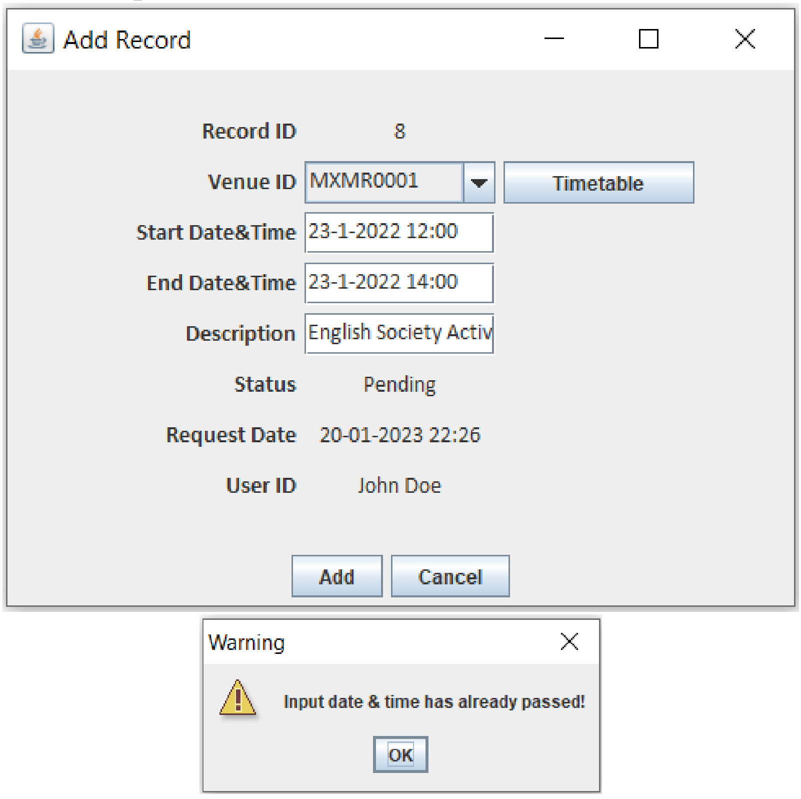
Record ID	Venue ID	Start Date&Time	End Date&Time	Description	Status	Request Date	User ID
2	MXMR0001	23-1-2023 15:00	23-1-2023 16:00	MMUSIC club activiti...	Denied	20-01-2023 13:20	John Doe
3	MXMR0001	22-1-2023 10:00	22-1-2023 11:00	Class self study	Pending	20-01-2023 20:56	John Doe
7	MXMR0002	22-1-2023 11:00	22-1-2023 12:00	Business Society A...	Approved	20-01-2023 20:51	John Doe

Click on any record to perform action

Figure 2.0.1 Student Main Menu

The main menu for students. The main menu displays all reservations records made under the user ID. Clicking “Log out” button returns to login page.

2.1 New Reservation Request



Add Record

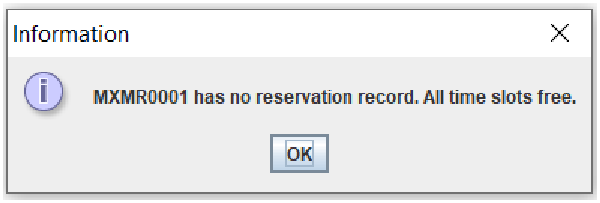
Record ID	8
Venue ID	MXMR0001
Start Date&Time	23-1-2022 12:00
End Date&Time	23-1-2022 14:00
Description	English Society Activ
Status	Pending
Request Date	20-01-2023 22:26
User ID	John Doe

Warning

Input date & time has already passed!

Figure 2.1.1 New reservation request

User is required to choose from a list of available venue, enter start Date & Time, end Date & Time, and description for the reservation. Timetable button displays current venue timetable. Input date & time needs to be larger than request time.

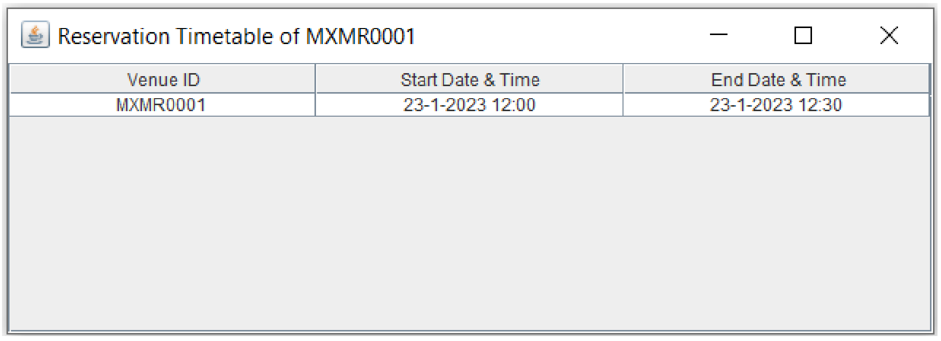


Information

MXMR0001 has no reservation record. All time slots free.

Figure 2.1.2 Checking venue timetable

A dialog pops up if venue has no reservation records.

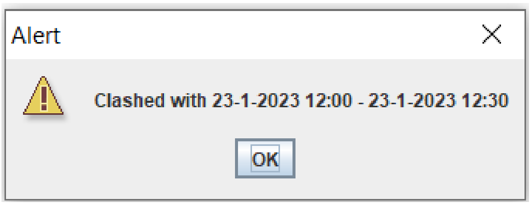


Reservation Timetable of MXMR0001

Venue ID	Start Date & Time	End Date & Time
MXMR0001	23-1-2023 12:00	23-1-2023 12:30

Figure 2.1.3 Checking venue timetable

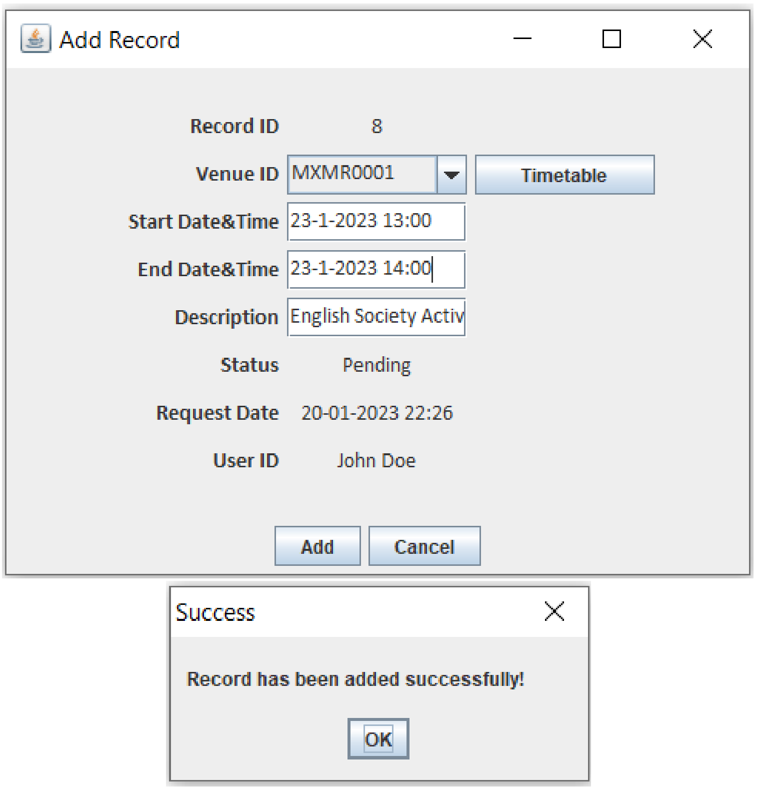
Reservation timetable is displayed if venue contains any booked timeslot.


Alert

Clashed with 23-1-2023 12:00 - 23-1-2023 12:30

Figure 2.1.4 Time clash between 2 reservations

Program checks if new reservation time clashes with any booked timeslots. New reservation is not saved if time clashed.


Add Record

Record ID	8
Venue ID	MXMR0001
Start Date&Time	23-1-2023 13:00
End Date&Time	23-1-2023 14:00
Description	English Society Activ
Status	Pending
Request Date	20-01-2023 22:26
User ID	John Doe

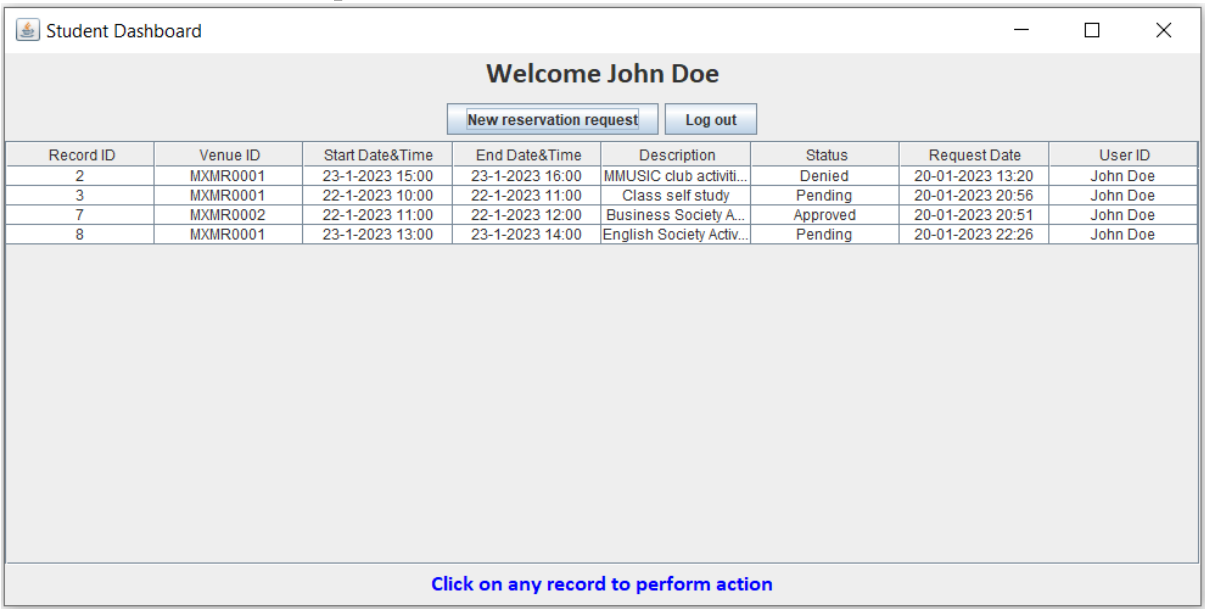
Success

Record has been added successfully!

Figure 2.1.5 Successful reservation

Reservation request is submitted if date & time is valid, and no time clashes occur.

2.2 Edit Reservation Request



Student Dashboard

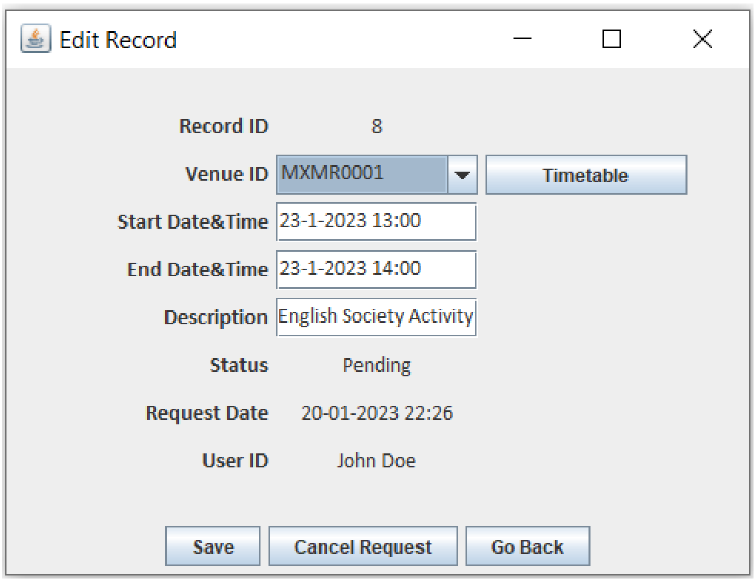
Welcome John Doe

Record ID	Venue ID	Start Date&Time	End Date&Time	Description	Status	Request Date	User ID
2	MXMR0001	23-1-2023 15:00	23-1-2023 16:00	MMUSIC club activiti...	Denied	20-01-2023 13:20	John Doe
3	MXMR0001	22-1-2023 10:00	22-1-2023 11:00	Class self study	Pending	20-01-2023 20:56	John Doe
7	MXMR0002	22-1-2023 11:00	22-1-2023 12:00	Business Society A...	Approved	20-01-2023 20:51	John Doe
8	MXMR0001	23-1-2023 13:00	23-1-2023 14:00	English Society Activ...	Pending	20-01-2023 22:26	John Doe

Click on any record to perform action

Figure 2.2.1 Edit Reservation Request

To edit a record, click on a record in the student dashboard.

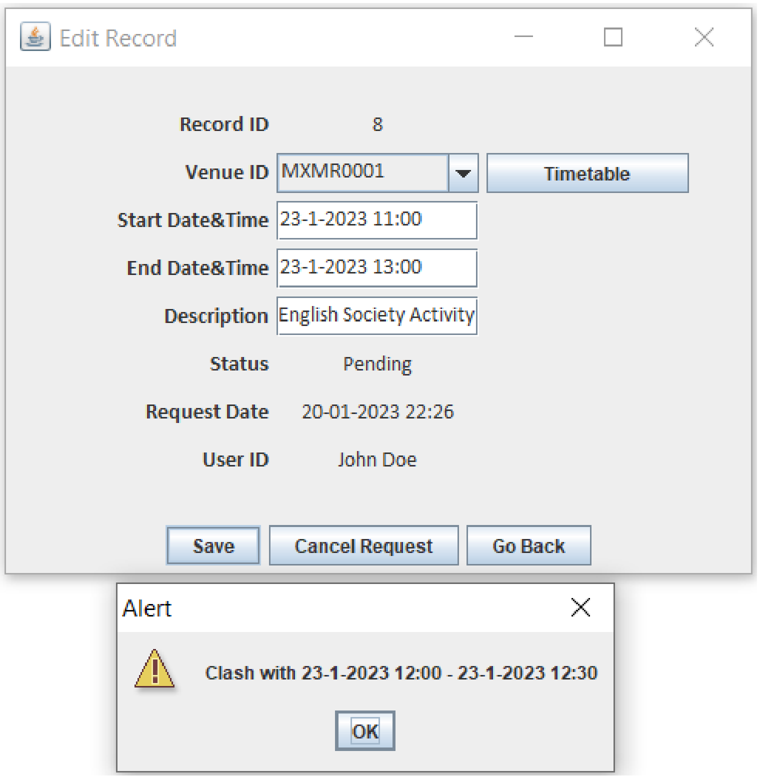


Edit Record

Record ID	8
Venue ID	MXMR0001
Start Date&Time	23-1-2023 13:00
End Date&Time	23-1-2023 14:00
Description	English Society Activity
Status	Pending
Request Date	20-01-2023 22:26
User ID	John Doe

Figure 2.2.2 Edit Reservation Request

The program displays current record information and allows the user to change reservation details.



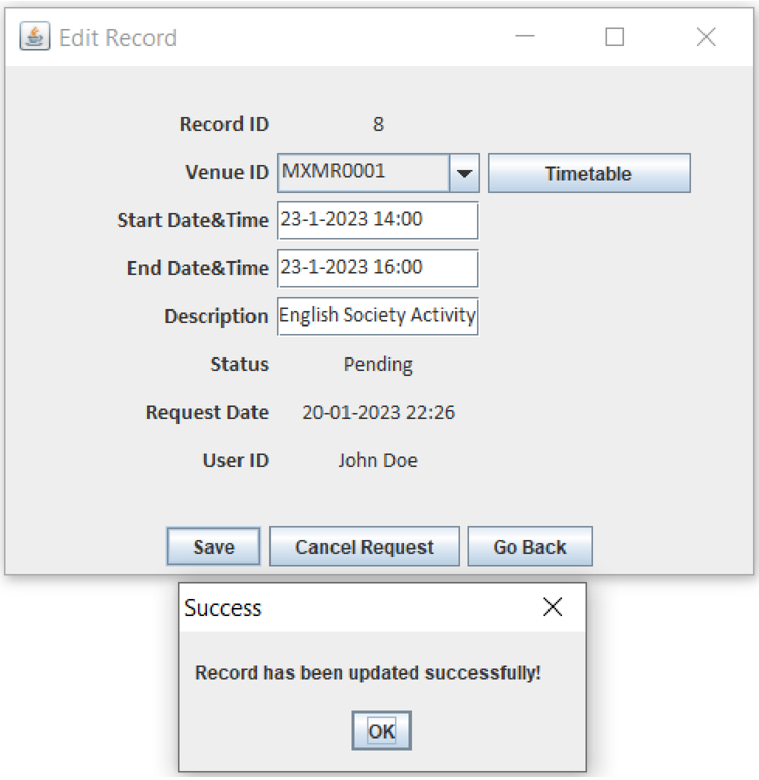
Record ID	8
Venue ID	MXMR0001
Start Date&Time	23-1-2023 11:00
End Date&Time	23-1-2023 13:00
Description	English Society Activity
Status	Pending
Request Date	20-01-2023 22:26
User ID	John Doe

Alert

Clash with 23-1-2023 12:00 - 23-1-2023 12:30

Figure 2.2.3 Edit reservation request

Start date & time and end date & time is validated and checked again for time clash.



Record ID	8
Venue ID	MXMR0001
Start Date&Time	23-1-2023 14:00
End Date&Time	23-1-2023 16:00
Description	English Society Activity
Status	Pending
Request Date	20-01-2023 22:26
User ID	John Doe

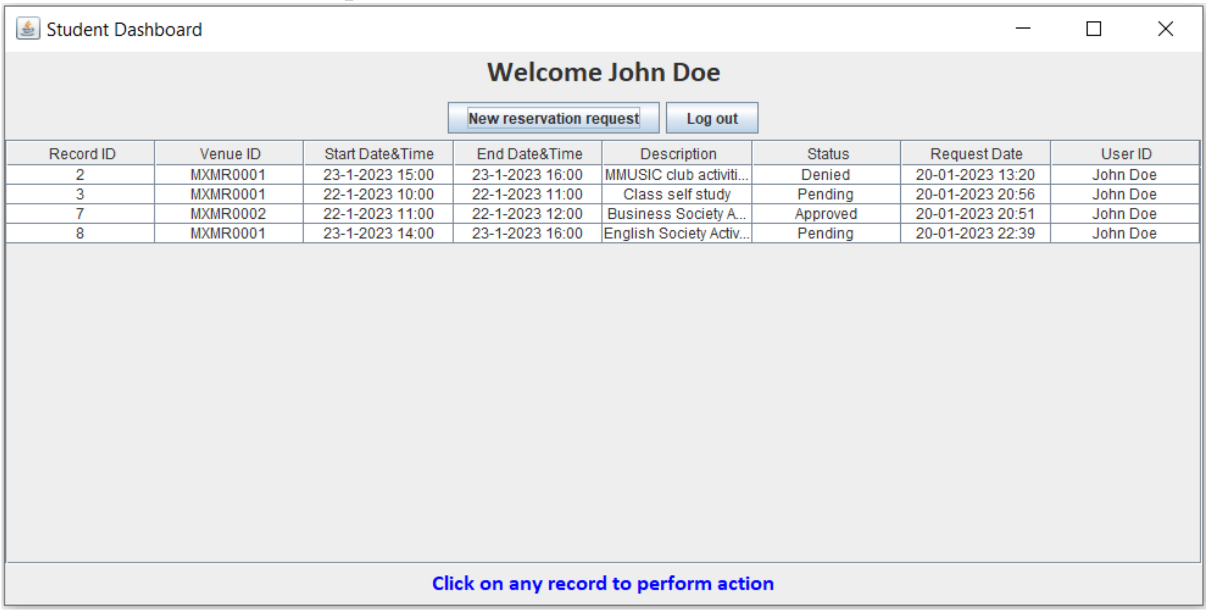
Success

Record has been updated successfully!

Figure 2.2.4 Edit reservation request

If no time clash occurs between input date & time with other timeslot, changes are saved. Record status will be switched to "Pending".

2.3 View Reservation Request


Student Dashboard

Welcome John Doe

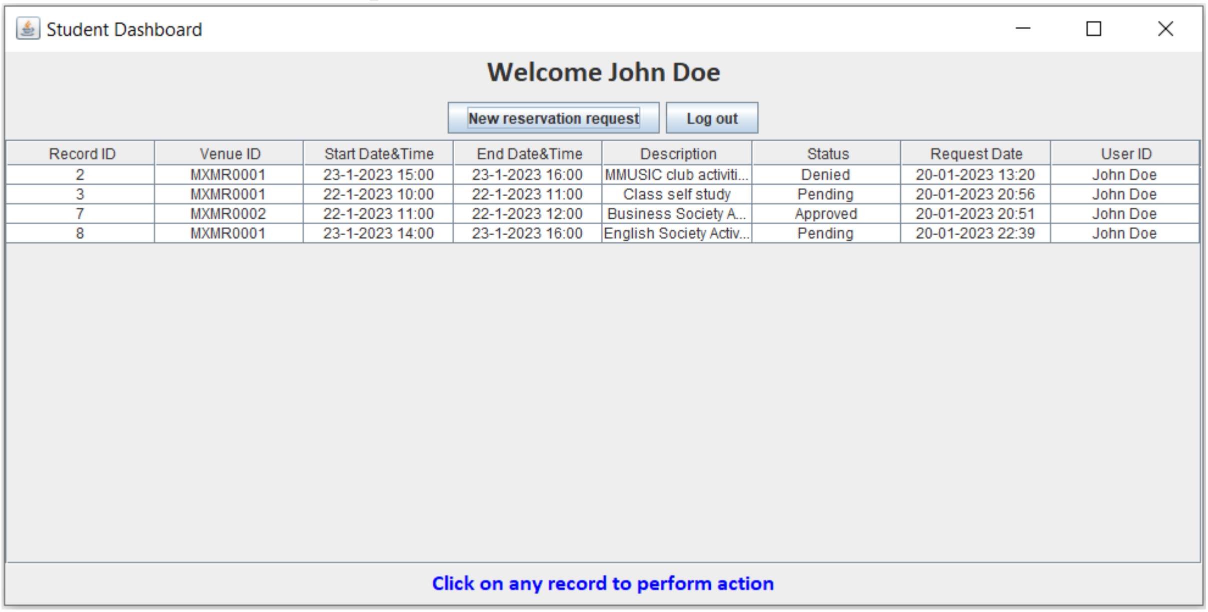
Record ID	Venue ID	Start Date&Time	End Date&Time	Description	Status	Request Date	User ID
2	MXMR0001	23-1-2023 15:00	23-1-2023 16:00	MMUSIC club activiti...	Denied	20-01-2023 13:20	John Doe
3	MXMR0001	22-1-2023 10:00	22-1-2023 11:00	Class self study	Pending	20-01-2023 20:56	John Doe
7	MXMR0002	22-1-2023 11:00	22-1-2023 12:00	Business Society A...	Approved	20-01-2023 20:51	John Doe
8	MXMR0001	23-1-2023 14:00	23-1-2023 16:00	English Society Activ...	Pending	20-01-2023 22:39	John Doe

Click on any record to perform action

Figure 2.3.1 View reservation request

The display table of reservation requests is merged with the main menu. Only reservations requests made under the User ID are shown.

2.4 Cancel Reservation Request


Student Dashboard

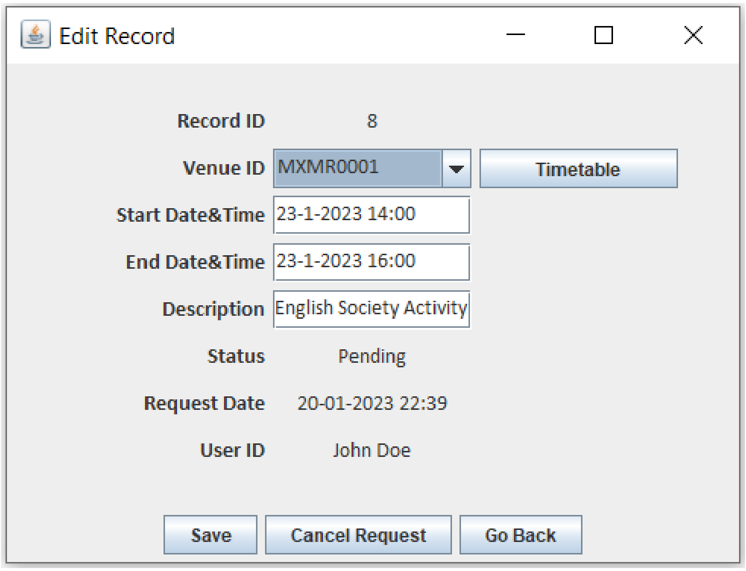
Welcome John Doe

Record ID	Venue ID	Start Date&Time	End Date&Time	Description	Status	Request Date	User ID
2	MXMR0001	23-1-2023 15:00	23-1-2023 16:00	MMUSIC club activiti...	Denied	20-01-2023 13:20	John Doe
3	MXMR0001	22-1-2023 10:00	22-1-2023 11:00	Class self study	Pending	20-01-2023 20:56	John Doe
7	MXMR0002	22-1-2023 11:00	22-1-2023 12:00	Business Society A...	Approved	20-01-2023 20:51	John Doe
8	MXMR0001	23-1-2023 14:00	23-1-2023 16:00	English Society Activ...	Pending	20-01-2023 22:39	John Doe

Click on any record to perform action

Figure 2.4.1 Cancel Reservation Request

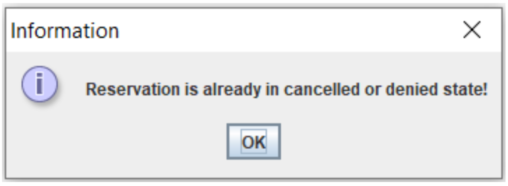
To cancel a reservation request, click on a record in the student dashboard.



Record ID 8
Venue ID MXMR0001
Start Date&Time 23-1-2023 14:00
End Date&Time 23-1-2023 16:00
Description English Society Activity
Status Pending
Request Date 20-01-2023 22:39
User ID John Doe

Figure 2.4.2 Cancel Reservation Request

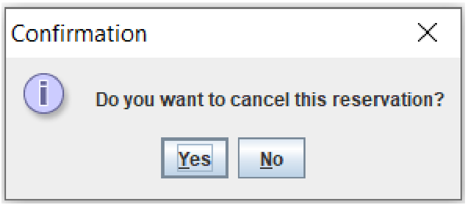
In the edit page, click on the "Cancel Request" button at the bottom of the screen.



Reservation is already in cancelled or denied state!

Figure 2.4.3 Cancel Reservation Request

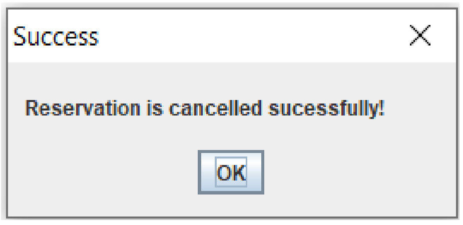
If the record status is already in "Cancelled" or "Denied" status, a dialog pops up and no changes will be made.



Do you want to cancel this reservation?

Figure 2.4.4 Cancel Confirmation

If the program is in "Approved" or "Pending" status, program confirms for cancellation.



Reservation is cancelled sucessfully!

Figure 2.4.5 Successful Cancellation

A dialog pops up if cancellation was successful. Record status is switched to "Cancelled".

3.0 Admin

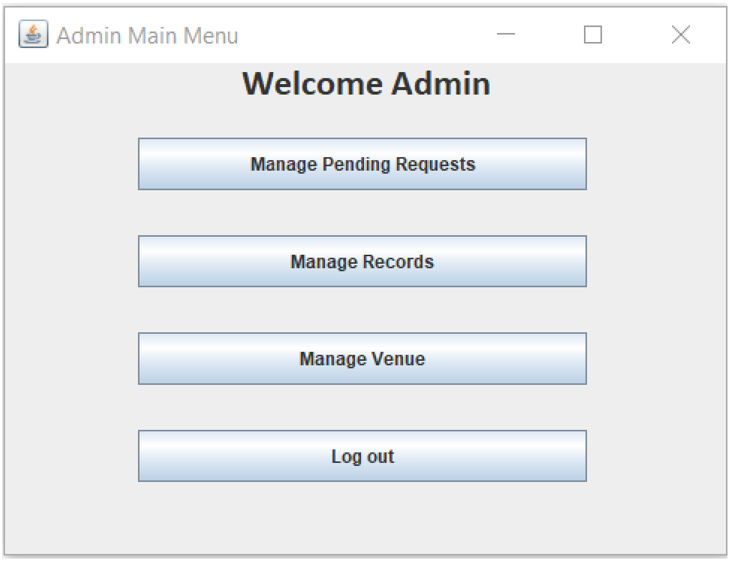
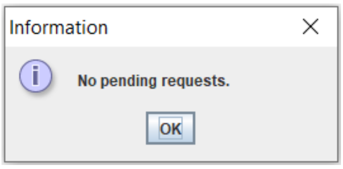


Figure 3.0.1 Admin Main Menu

The main menu for admin. Clicking “Log out” button returns to login page.

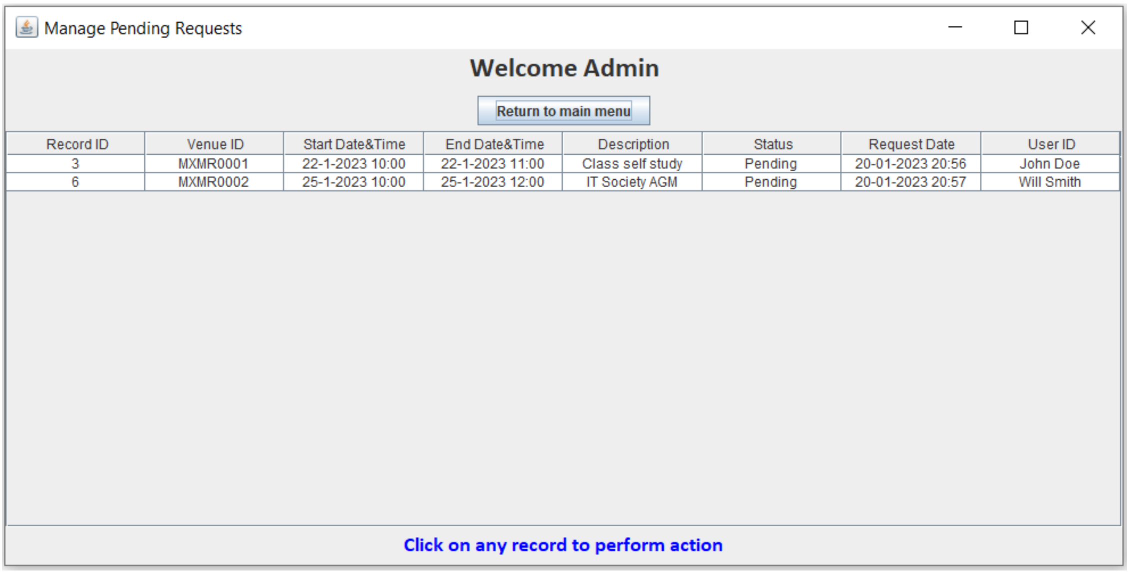
3.1 Manage Pending Requests



No pending requests.

Figure 3.1.1 Manage Pending Request

If there is no pending request available in the system, a dialog pops up.



Record ID	Venue ID	Start Date&Time	End Date&Time	Description	Status	Request Date	User ID
3	MXMR0001	22-1-2023 10:00	22-1-2023 11:00	Class self study	Pending	20-01-2023 20:56	John Doe
6	MXMR0002	25-1-2023 10:00	25-1-2023 12:00	IT Society AGM	Pending	20-01-2023 20:57	Will Smith

Click on any record to perform action

Figure 3.1.2 Process Pending Request

If there are pending requests available in the system, a table of pending requests is displayed. To process a request, click on any record.

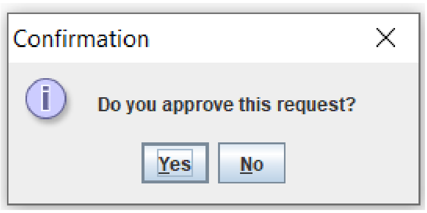


Figure 3.1.3 Approve/Deny Request

Admin may choose to Approve or Deny the request. Close the dialog box to cancel process.

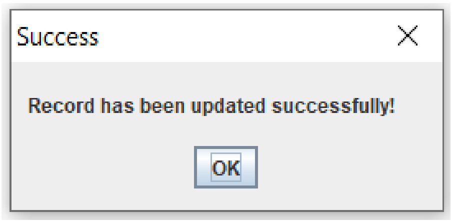
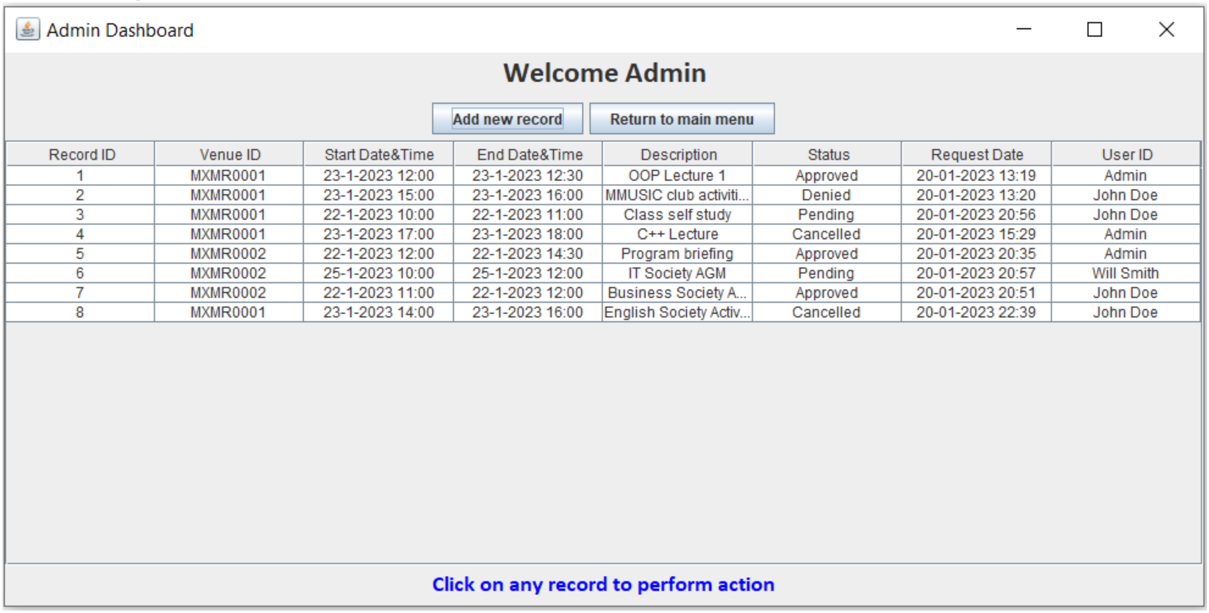


Figure 3.1.4 Successful Operation

A dialog pops up if Approval/Denial is performed successfully. The frame is then refreshed to update the display list.

3.2 Manage Records

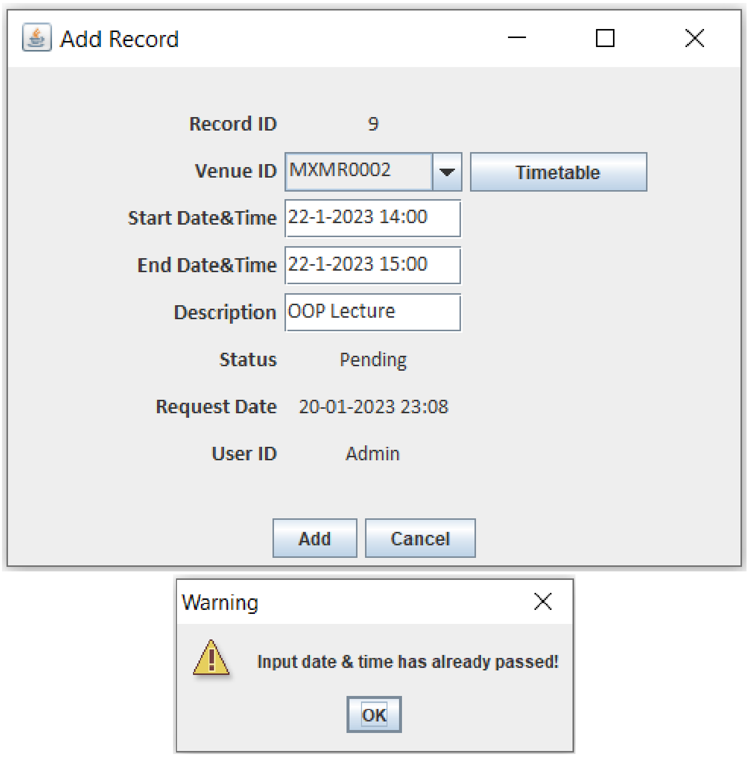


Record ID	Venue ID	Start Date&Time	End Date&Time	Description	Status	Request Date	User ID
1	MXMR0001	23-1-2023 12:00	23-1-2023 12:30	OOP Lecture 1	Approved	20-01-2023 13:19	Admin
2	MXMR0001	23-1-2023 15:00	23-1-2023 16:00	MMUSIC club activiti...	Denied	20-01-2023 13:20	John Doe
3	MXMR0001	22-1-2023 10:00	22-1-2023 11:00	Class self study	Pending	20-01-2023 20:56	John Doe
4	MXMR0001	23-1-2023 17:00	23-1-2023 18:00	C++ Lecture	Cancelled	20-01-2023 15:29	Admin
5	MXMR0002	22-1-2023 12:00	22-1-2023 14:30	Program briefing	Approved	20-01-2023 20:35	Admin
6	MXMR0002	25-1-2023 10:00	25-1-2023 12:00	IT Society AGM	Pending	20-01-2023 20:57	Will Smith
7	MXMR0002	22-1-2023 11:00	22-1-2023 12:00	Business Society A...	Approved	20-01-2023 20:51	John Doe
8	MXMR0001	23-1-2023 14:00	23-1-2023 16:00	English Society Activ...	Cancelled	20-01-2023 22:39	John Doe

Figure 3.2.0.1 Records Dashboard

This is the admin dashboard of Manage Records. The dashboard displays all available records in the system.

3.2.1 Add New Record



Add Record

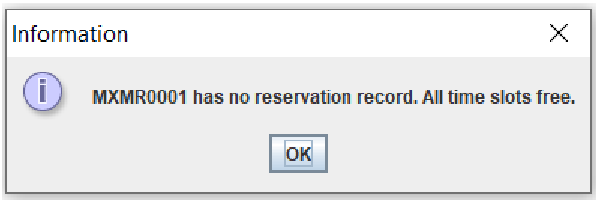
Record ID	9
Venue ID	MXMR0002
Start Date&Time	22-1-2023 14:00
End Date&Time	22-1-2023 15:00
Description	OOP Lecture
Status	Pending
Request Date	20-01-2023 23:08
User ID	Admin

Warning

Input date & time has already passed!

Figure 3.2.1.1 Add New Record

User is required to choose from a list of available venue, enter Start Date & Time, end Date & Time, and description for the reservation. Timetable button displays current venue timetable. Input date & time needs to be larger than request date & time.

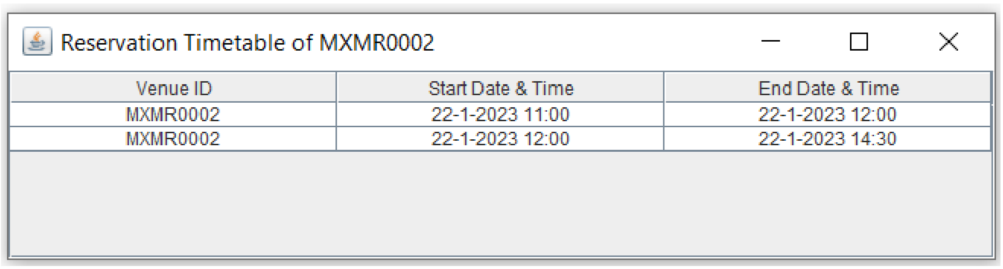


Information

MXMR0001 has no reservation record. All time slots free.

Figure 3.2.1.2 Checking Venue Timetable

A dialog pops up if venue has no reservation records.

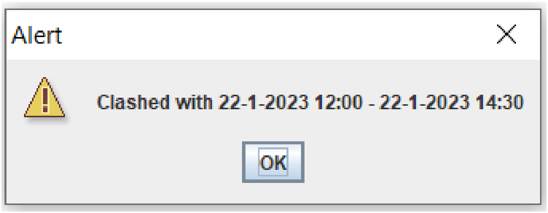


Reservation Timetable of MXMR0002

Venue ID	Start Date & Time	End Date & Time
MXMR0002	22-1-2023 11:00	22-1-2023 12:00
MXMR0002	22-1-2023 12:00	22-1-2023 14:30

Figure 3.2.1.3 Venue Timetable

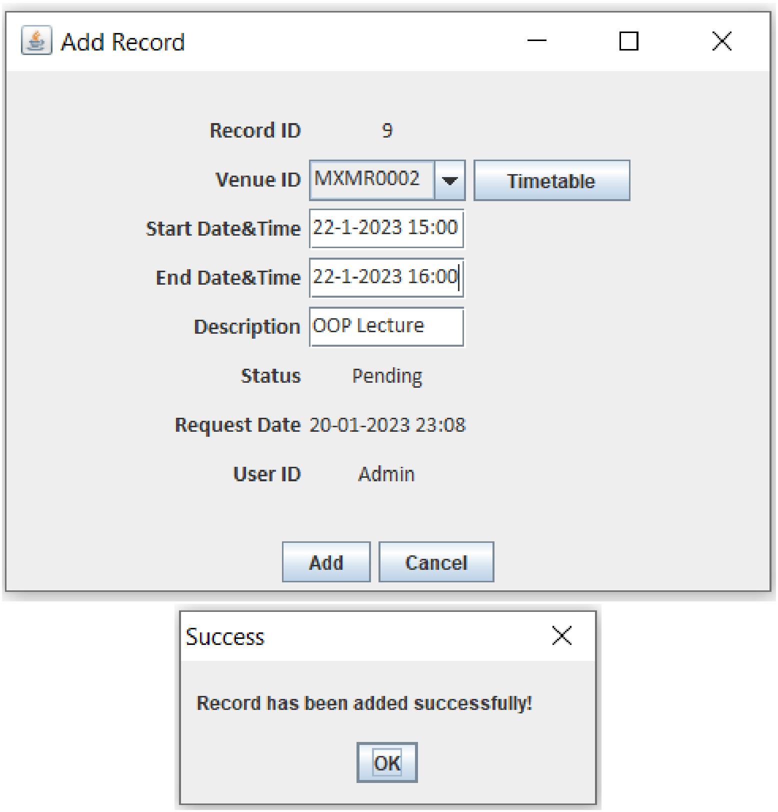
Reservation timetable is displayed if venue contains any booked timeslot.


Alert

Clashed with 22-1-2023 12:00 - 22-1-2023 14:30

Figure 3.2.1.4 Time Clash between 2 Records

Program checks if new reservation time clashes with any booked timeslots. New reservation is not saved if time clashed.


Add Record

Record ID	9
Venue ID	MXMR0002
Start Date&Time	22-1-2023 15:00
End Date&Time	22-1-2023 16:00
Description	OOP Lecture
Status	Pending
Request Date	20-01-2023 23:08
User ID	Admin

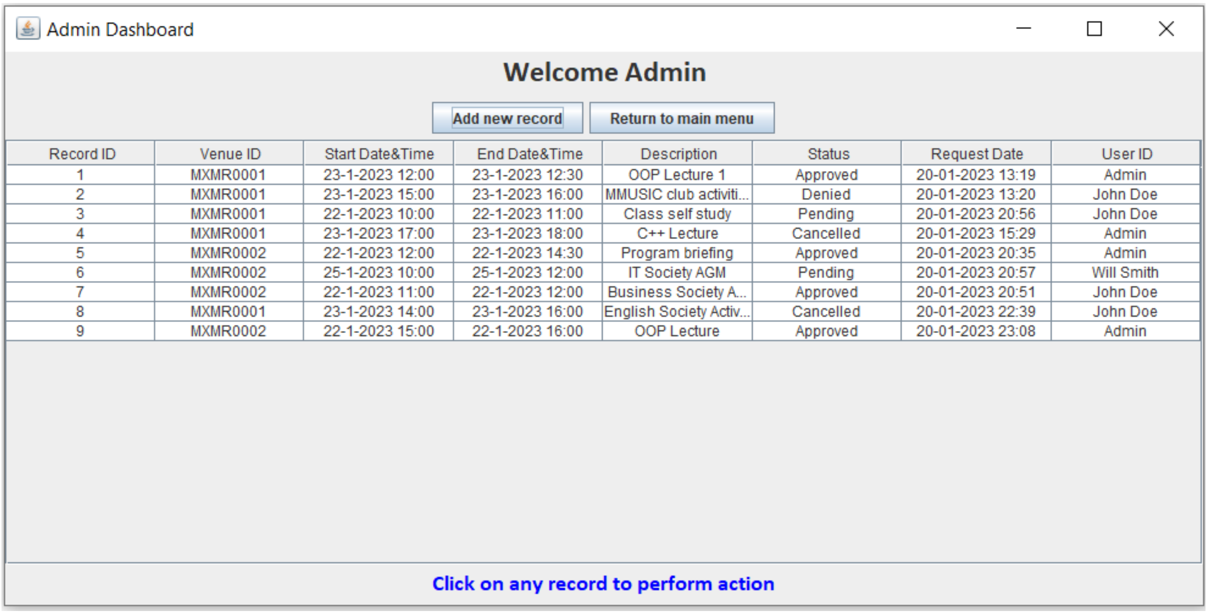
Success

Record has been added successfully!

Figure 3.2.1.5 Add Record Successful

Reservation request is submitted if date & time is valid, and no time clashes occur. Status of all record submitted by admin will be saved as “Approved”.

3.2.2 Edit record


Admin Dashboard

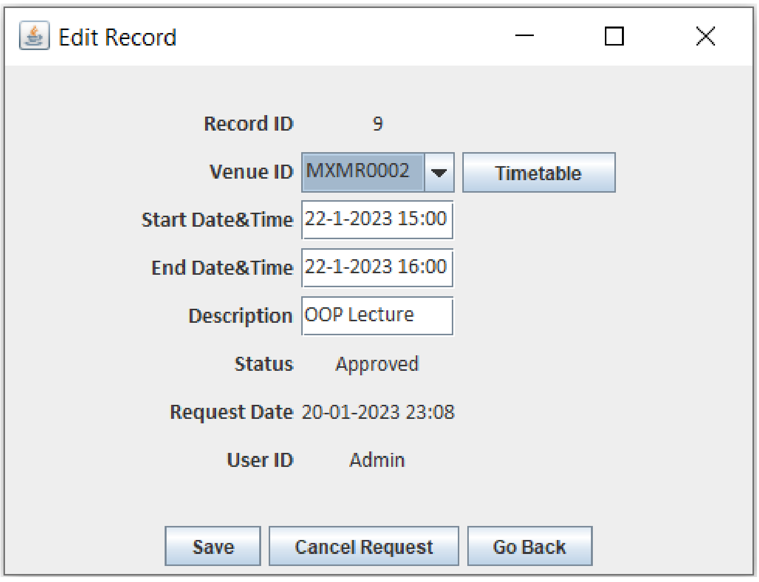
Welcome Admin

Record ID	Venue ID	Start Date&Time	End Date&Time	Description	Status	Request Date	User ID
1	MXMR0001	23-1-2023 12:00	23-1-2023 12:30	OOP Lecture 1	Approved	20-01-2023 13:19	Admin
2	MXMR0001	23-1-2023 15:00	23-1-2023 16:00	MMUSIC club activiti...	Denied	20-01-2023 13:20	John Doe
3	MXMR0001	22-1-2023 10:00	22-1-2023 11:00	Class self study	Pending	20-01-2023 20:56	John Doe
4	MXMR0001	23-1-2023 17:00	23-1-2023 18:00	C++ Lecture	Cancelled	20-01-2023 15:29	Admin
5	MXMR0002	22-1-2023 12:00	22-1-2023 14:30	Program briefing	Approved	20-01-2023 20:35	Admin
6	MXMR0002	25-1-2023 10:00	25-1-2023 12:00	IT Society AGM	Pending	20-01-2023 20:57	Will Smith
7	MXMR0002	22-1-2023 11:00	22-1-2023 12:00	Business Society A...	Approved	20-01-2023 20:51	John Doe
8	MXMR0001	23-1-2023 14:00	23-1-2023 16:00	English Society Activ...	Cancelled	20-01-2023 22:39	John Doe
9	MXMR0002	22-1-2023 15:00	22-1-2023 16:00	OOP Lecture	Approved	20-01-2023 23:08	Admin

Click on any record to perform action

Figure 3.2.2.1 Edit Record

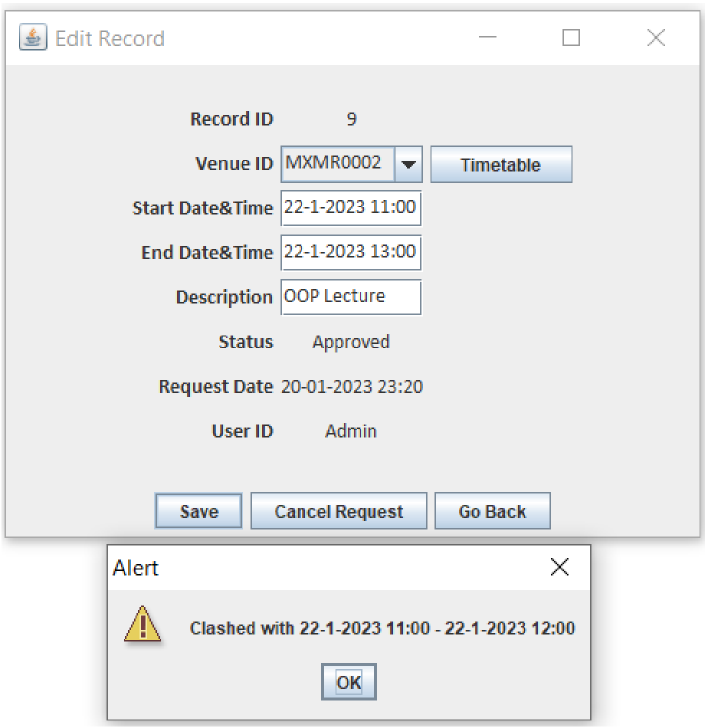
To edit a record, click on any record in the admin dashboard. Admin can access and edit all available records in the system.


Edit Record

Record ID	9
Venue ID	MXMR0002
Start Date&Time	22-1-2023 15:00
End Date&Time	22-1-2023 16:00
Description	OOP Lecture
Status	Approved
Request Date	20-01-2023 23:08
User ID	Admin

Figure 3.2.2.2 Edit Record

The program displays current record information and allows admin to change reservation details.


Edit Record

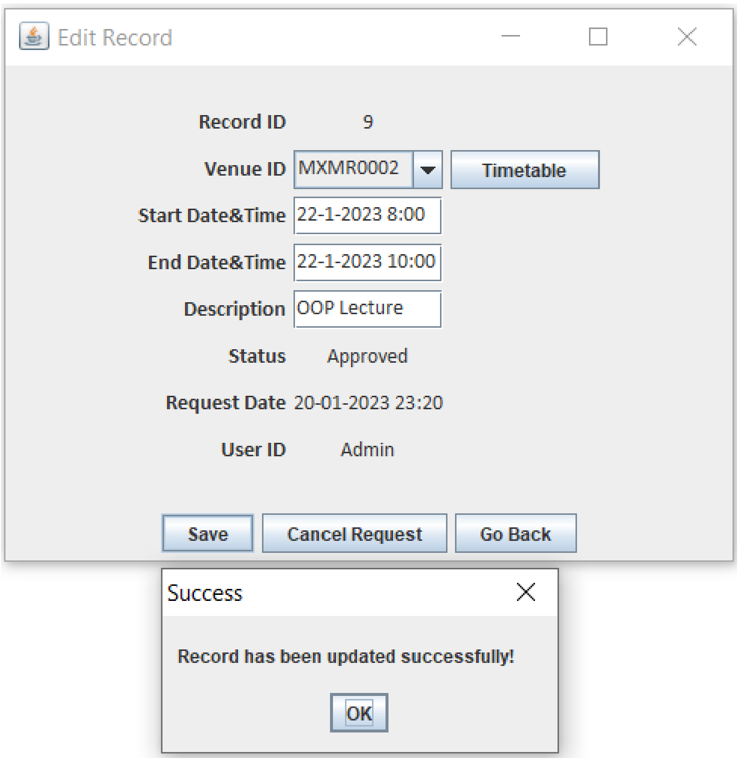
Record ID	9
Venue ID	MXMR0002
Start Date&Time	22-1-2023 11:00
End Date&Time	22-1-2023 13:00
Description	OOP Lecture
Status	Approved
Request Date	20-01-2023 23:20
User ID	Admin

Alert

Clashed with 22-1-2023 11:00 - 22-1-2023 12:00

Figure 3.2.2.3 Edit Record

Start date & time and end date & time is validated and checked again for time clash.


Edit Record

Record ID	9
Venue ID	MXMR0002
Start Date&Time	22-1-2023 8:00
End Date&Time	22-1-2023 10:00
Description	OOP Lecture
Status	Approved
Request Date	20-01-2023 23:20
User ID	Admin

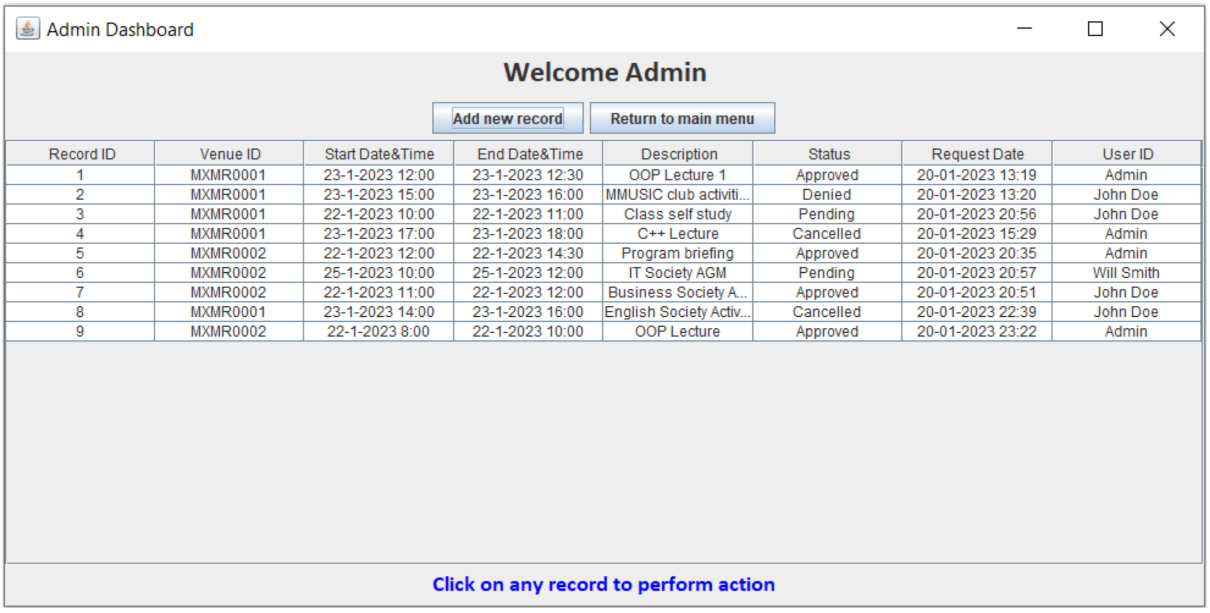
Success

Record has been updated successfully!

Figure 3.2.2.4 Edit Record

If no time clash occurs between input date & time with other timeslot, changes are saved. Record status will be saved as "Approved" regardless of its original status.

3.2.3 View all records

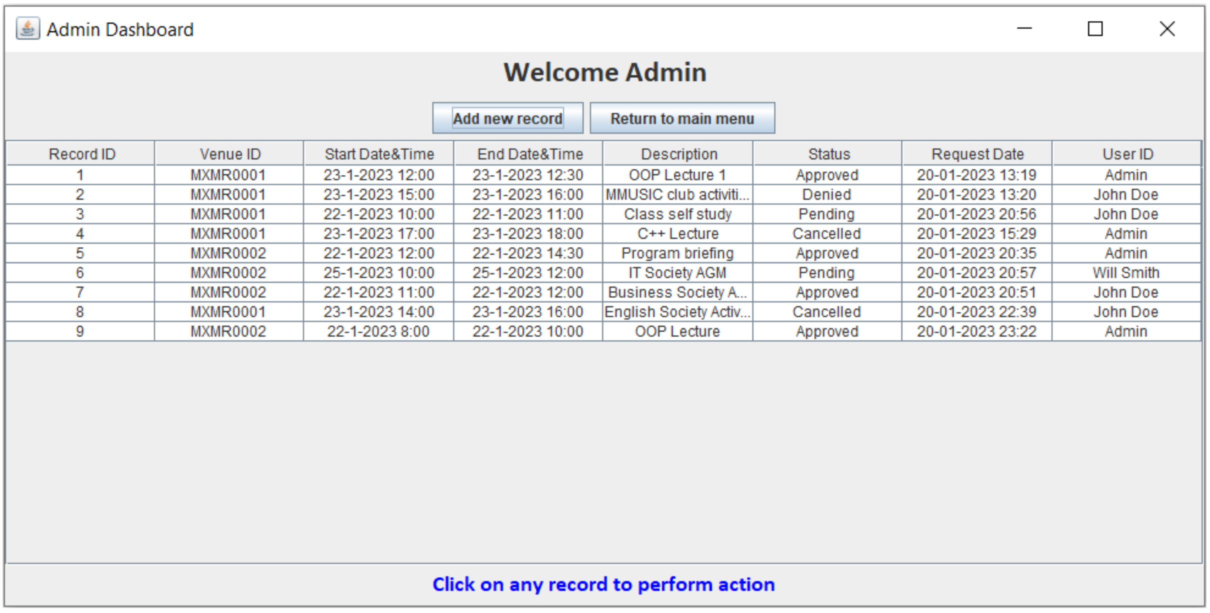


Record ID	Venue ID	Start Date&Time	End Date&Time	Description	Status	Request Date	User ID
1	MXMR0001	23-1-2023 12:00	23-1-2023 12:30	OOP Lecture 1	Approved	20-01-2023 13:19	Admin
2	MXMR0001	23-1-2023 15:00	23-1-2023 16:00	MMUSIC club activiti...	Denied	20-01-2023 13:20	John Doe
3	MXMR0001	22-1-2023 10:00	22-1-2023 11:00	Class self study	Pending	20-01-2023 20:56	John Doe
4	MXMR0001	23-1-2023 17:00	23-1-2023 18:00	C++ Lecture	Cancelled	20-01-2023 15:29	Admin
5	MXMR0002	22-1-2023 12:00	22-1-2023 14:30	Program briefing	Approved	20-01-2023 20:35	Admin
6	MXMR0002	25-1-2023 10:00	25-1-2023 12:00	IT Society AGM	Pending	20-01-2023 20:57	Will Smith
7	MXMR0002	22-1-2023 11:00	22-1-2023 12:00	Business Society A...	Approved	20-01-2023 20:51	John Doe
8	MXMR0001	23-1-2023 14:00	23-1-2023 16:00	English Society Activ...	Cancelled	20-01-2023 22:39	John Doe
9	MXMR0002	22-1-2023 8:00	22-1-2023 10:00	OOP Lecture	Approved	20-01-2023 23:22	Admin

Figure 3.2.3.1 View records

The display table of reservation records is merged with the admin dashboard. Admin may access and view records made by all users in the system.

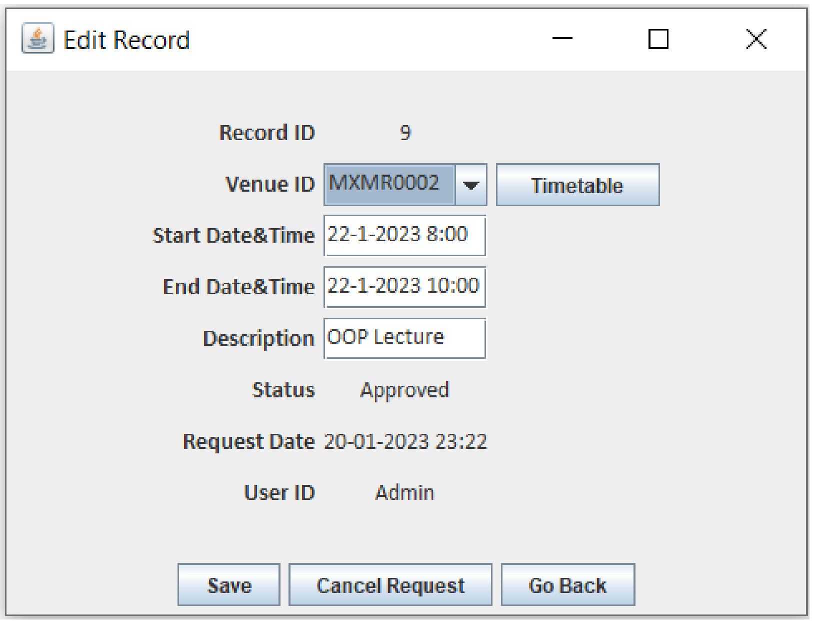
3.2.4 Cancel a record



Record ID	Venue ID	Start Date&Time	End Date&Time	Description	Status	Request Date	User ID
1	MXMR0001	23-1-2023 12:00	23-1-2023 12:30	OOP Lecture 1	Approved	20-01-2023 13:19	Admin
2	MXMR0001	23-1-2023 15:00	23-1-2023 16:00	MMUSIC club activiti...	Denied	20-01-2023 13:20	John Doe
3	MXMR0001	22-1-2023 10:00	22-1-2023 11:00	Class self study	Pending	20-01-2023 20:56	John Doe
4	MXMR0001	23-1-2023 17:00	23-1-2023 18:00	C++ Lecture	Cancelled	20-01-2023 15:29	Admin
5	MXMR0002	22-1-2023 12:00	22-1-2023 14:30	Program briefing	Approved	20-01-2023 20:35	Admin
6	MXMR0002	25-1-2023 10:00	25-1-2023 12:00	IT Society AGM	Pending	20-01-2023 20:57	Will Smith
7	MXMR0002	22-1-2023 11:00	22-1-2023 12:00	Business Society A...	Approved	20-01-2023 20:51	John Doe
8	MXMR0001	23-1-2023 14:00	23-1-2023 16:00	English Society Activ...	Cancelled	20-01-2023 22:39	John Doe
9	MXMR0002	22-1-2023 8:00	22-1-2023 10:00	OOP Lecture	Approved	20-01-2023 23:22	Admin

Figure 3.2.4.1 Cancel Record

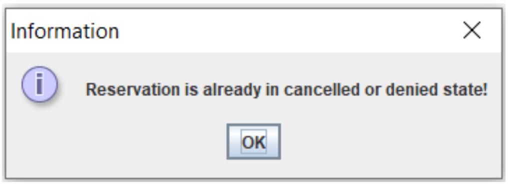
To cancel a record, click on any records in the admin dashboard.



Record ID 9
Venue ID MXMR0002
Start Date&Time 22-1-2023 8:00
End Date&Time 22-1-2023 10:00
Description OOP Lecture
Status Approved
Request Date 20-01-2023 23:22
User ID Admin

Figure 3.2.4.2 Cancel Record

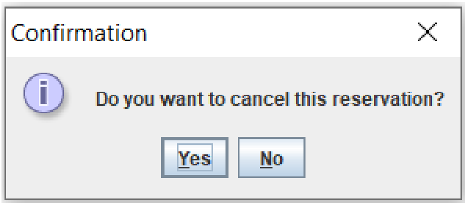
In the edit page, click on the "Cancel Request" button at the bottom of the screen.



Reservation is already in cancelled or denied state!

Figure 3.2.4.3 Cancel Reservation Request

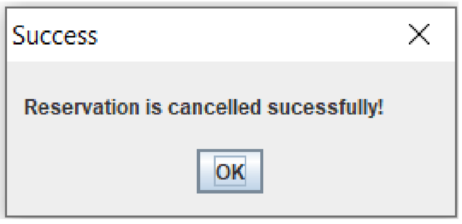
If the record status is already in "Cancelled" or "Denied" status, a dialog pops up and no changes will be made.



Do you want to cancel this reservation?

Figure 3.2.4.4 Cancel Confirmation

If the program is in "Approved" or "Pending" status, program confirms for cancellation.

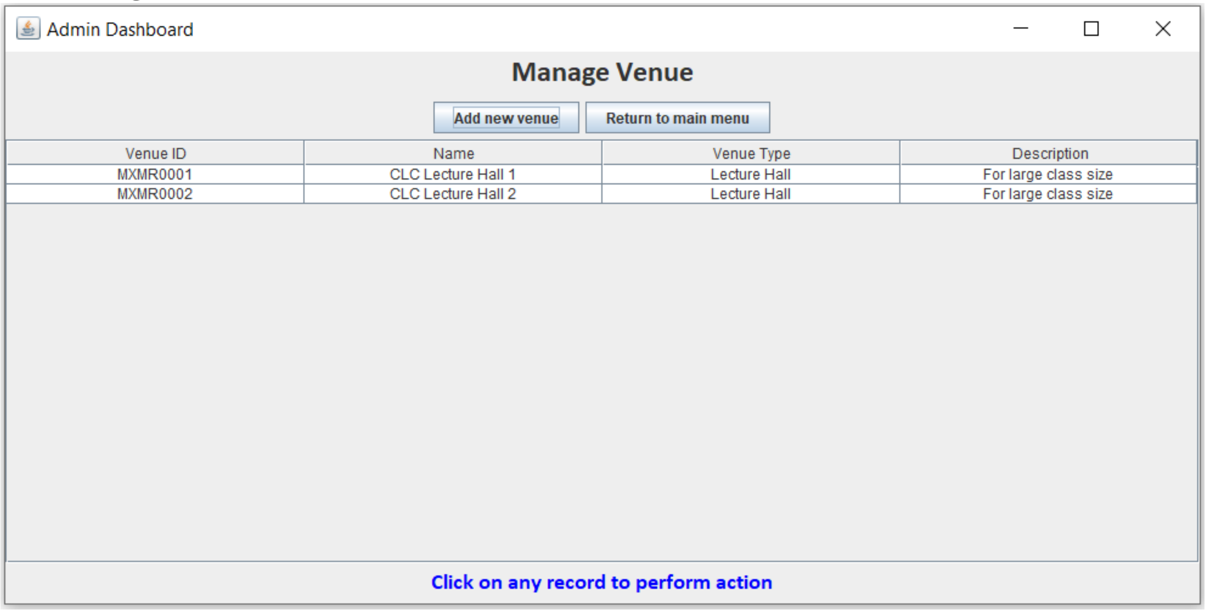


Reservation is cancelled sucessfully!

Figure 3.2.4.5 Successful Cancellation

A dialog pops up if cancellation was successful. Record status is switched to "Cancelled".

3.3 Manage Venue

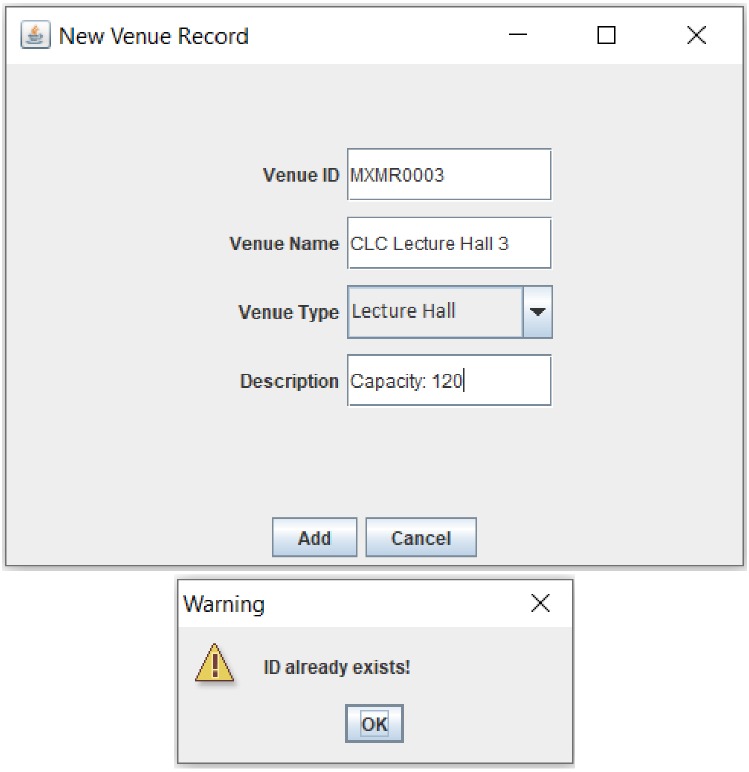


Venue ID	Name	Venue Type	Description
MXMR0001	CLC Lecture Hall 1	Lecture Hall	For large class size
MXMR0002	CLC Lecture Hall 2	Lecture Hall	For large class size

Figure 3.3.0.1 Venue Dashboard

This is the admin dashboard of Manage Venue. The dashboard displays all available venue records in the system.

3.3.1 Add New Venue

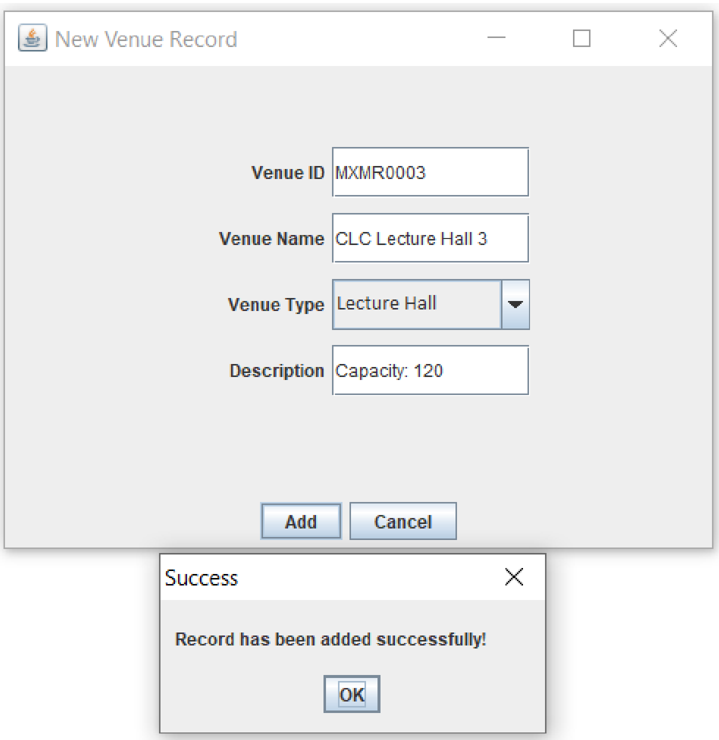


Venue ID: MXMR0003
Venue Name: CLC Lecture Hall 3
Venue Type: Lecture Hall
Description: Capacity: 120

ID already exists!

Figure 3.3.1.1 Add Venue

User is asked to enter venue information. Venue ID cannot be an existing venue ID.


New Venue Record

Venue ID: MXMR0003
Venue Name: CLC Lecture Hall 3
Venue Type: Lecture Hall
Description: Capacity: 120

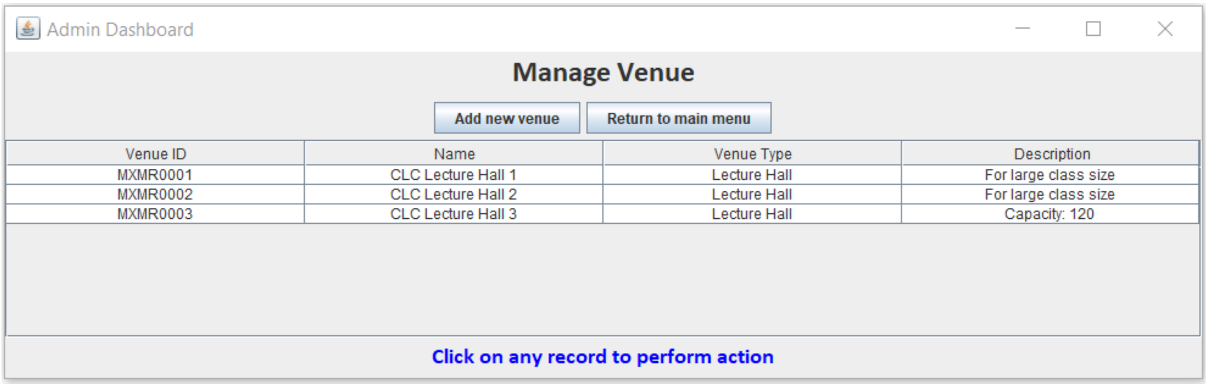
Success

Record has been added successfully!

Figure 3.3.1.2 Add Venue

A dialog pops up if the venue record is successfully added.

3.3.2 Edit Venue Information


Admin Dashboard

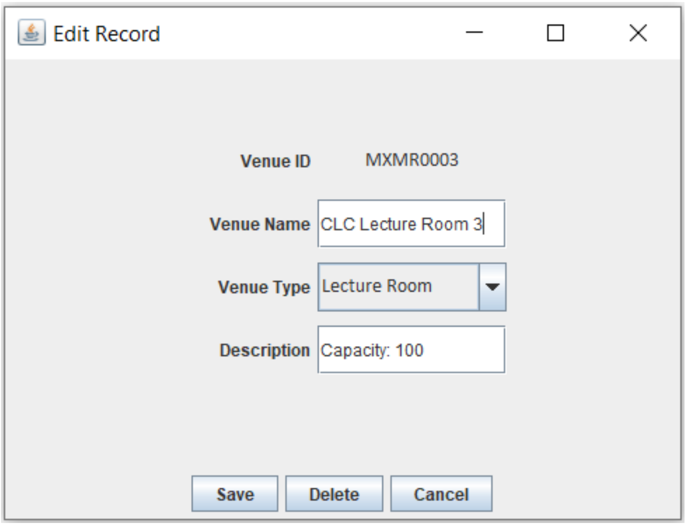
Manage Venue

Venue ID	Name	Venue Type	Description
MXMR0001	CLC Lecture Hall 1	Lecture Hall	For large class size
MXMR0002	CLC Lecture Hall 2	Lecture Hall	For large class size
MXMR0003	CLC Lecture Hall 3	Lecture Hall	Capacity: 120

Click on any record to perform action

Figure 3.3.2.1 Edit Venue

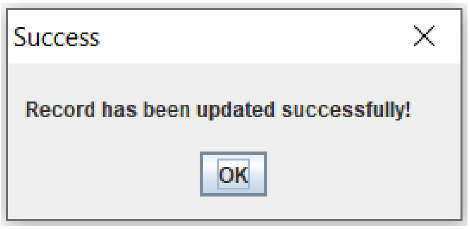
To edit a venue record, click on any record on the admin dashboard.


Edit Record

Venue ID	MXMR0003
Venue Name	CLC Lecture Room 3
Venue Type	Lecture Room
Description	Capacity: 100

Figure 3.3.2.2 Edit Venue

The program displays current venue information and allows admin to change venue information.

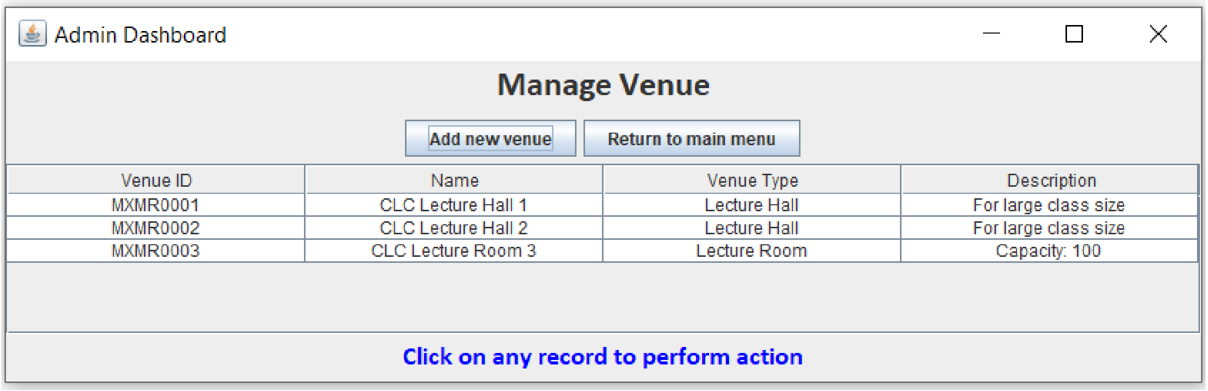

Success

Record has been updated successfully!

Figure 3.3.2.3 Successful Edit

A dialog pops up if the venue information has been successful edited.

3.3.3 View All Venue


Admin Dashboard

Manage Venue

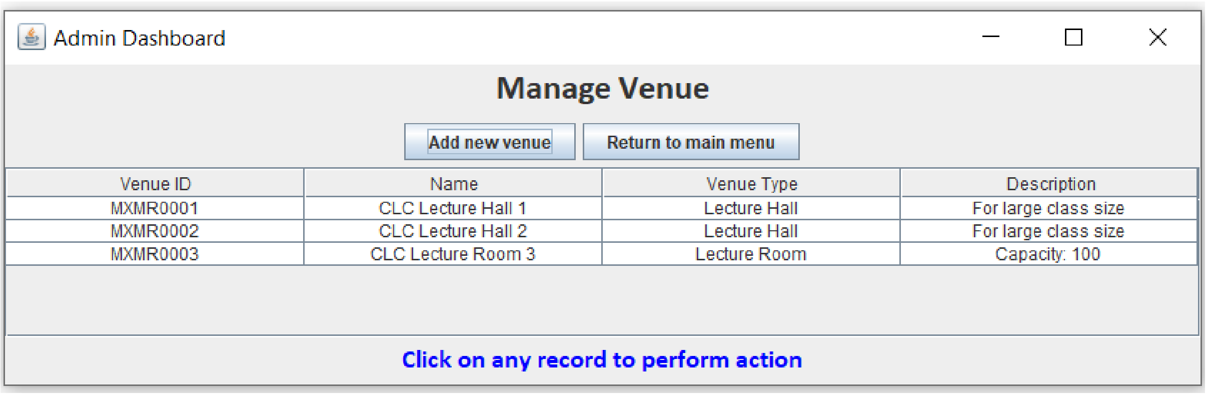
Venue ID	Name	Venue Type	Description
MXMR0001	CLC Lecture Hall 1	Lecture Hall	For large class size
MXMR0002	CLC Lecture Hall 2	Lecture Hall	For large class size
MXMR0003	CLC Lecture Room 3	Lecture Room	Capacity: 100

Click on any record to perform action

Figure 3.3.3.1 View Venue

The display table of all venue records is merged with the dashboard.

3.3.4 Delete Venue

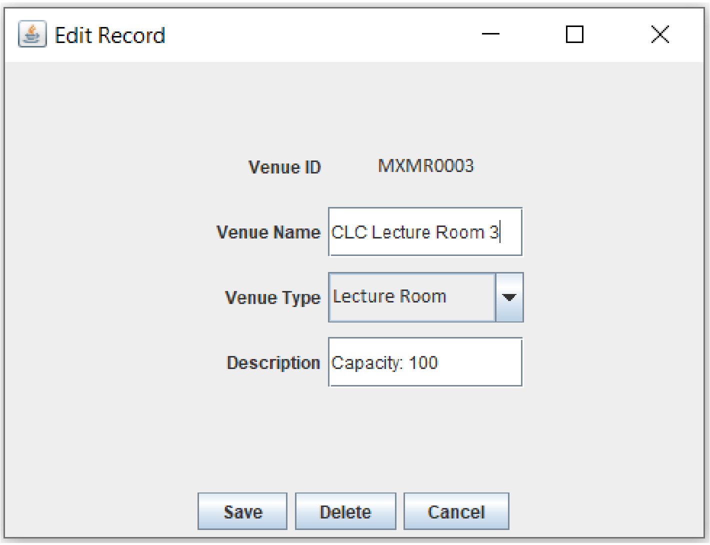


Venue ID	Name	Venue Type	Description
MXMR0001	CLC Lecture Hall 1	Lecture Hall	For large class size
MXMR0002	CLC Lecture Hall 2	Lecture Hall	For large class size
MXMR0003	CLC Lecture Room 3	Lecture Room	Capacity: 100

Click on any record to perform action

Figure 3.3.4.1 Delete Record

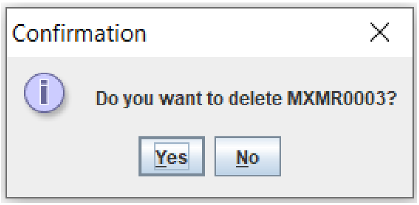
To delete a record, click on any records in the admin dashboard.



Venue ID MXMR0003
Venue Name CLC Lecture Room 3
Venue Type Lecture Room
Description Capacity: 100

Figure 3.3.4.2 Delete Record

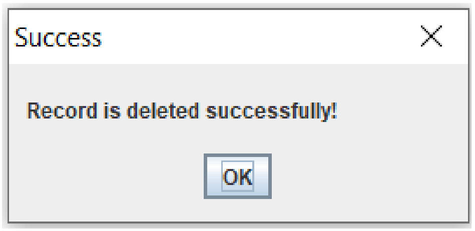
In the edit page, click on the “Delete” button.



Do you want to delete MXMR0003?

Figure 3.3.4.3 Delete Confirmation

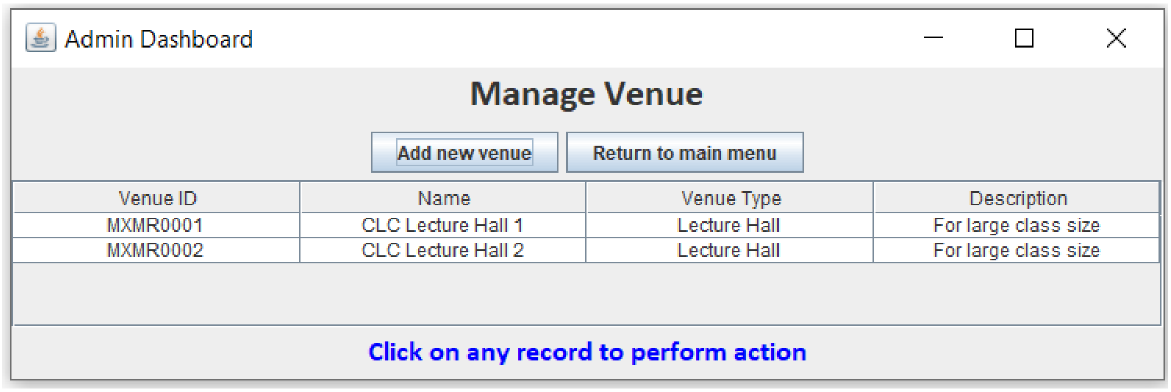
The program asks for delete confirmation.



Record is deleted successfully!

Figure 3.3.4.4 Successful Deletion

A dialog pops up if record is successfully deleted.



Manage Venue

Venue ID	Name	Venue Type	Description
MXMR0001	CLC Lecture Hall 1	Lecture Hall	For large class size
MXMR0002	CLC Lecture Hall 2	Lecture Hall	For large class size

Click on any record to perform action

Figure 3.3.4.5 Deletion Output

The record is now completely deleted from the system.

Reference

Java ActionListener - javatpoint. (n.d.). www.javatpoint.com. https://www.javatpoint.com/java-actionlistener

Java BoxLayout - javatpoint. (n.d.). www.javatpoint.com. https://www.javatpoint.com/BoxLayout

Java JButton - javatpoint. (n.d.). www.javatpoint.com. https://www.javatpoint.com/java-jbutton

Java JFrame - javatpoint. (n.d.). www.javatpoint.com. https://www.javatpoint.com/java-jframe

Java JTable - javatpoint. (n.d.). www.javatpoint.com. https://www.javatpoint.com/java-jtable

Java Layout Manager - javatpoint. (n.d.). www.javatpoint.com. https://www.javatpoint.com/java-layout-manager

JComboBox (Java Platform SE 6). (2015, November 19). https://docs.oracle.com/javase/6/docs/api/javax/swing/JComboBox.html

JOptionPane (Java Platform SE 6). (2015, November 19). https://docs.oracle.com/javase/6/docs/api/javax/swing/JOptionPane.html

JTextField (Java Platform SE 7). (2020, June 24). https://docs.oracle.com/javase/7/docs/api/javax/swing/JTextField.html

Minh, N. H., & Minh, N. H. (n.d.). *JFrame basic tutorial and examples*. https://www.codejava.net/java-se/swing/jframe-basic-tutorial-and-examples

Mv, T. (2020, November 23). *Java String to Int – How to Convert a String to an Integer*. freeCodeCamp.org. https://www.freecodecamp.org/news/java-string-to-int-how-to-convert-a-string-to-an-integer/

Appendix

Source Code

In this coding project, the necessary files include “UVMS.java”, “User.java”, “Student.java”, “Staff.java”, “Venue.java”, “GUI.java”, and “login.txt”. The source code is a mixture of frontend graphical user interface code and backend system code.

login.txt

```
admin admin
```

UVMS.java

```
import java.awt.*;
import java.util.*;
import org.omg.CORBA.Request;
import java.io.*;
import java.nio.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.JTable;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableModel;

/*This is the main file of the system.
 * Compile and run to start the GUI program.
 */
public class UVMS{
    static GUI gui = new GUI();
    public static void main(String args[]){
        //Title of the Program
        final JFrame f = gui.createFrame("University Venue Management 
System");
        JLabel l1 = gui.createHeadingLabel("University Venue Management 
System");
        f.add(l1, BorderLayout.NORTH);

        final JButton stuLogin = new JButton("Student Login");
        final JButton adminLogin = new JButton("Admin Login");
        final JButton exitProg = new JButton("Exit Program");

        //Create a panel to hold the buttons
        JPanel buttonPanel = gui.createVertPanel(3, 1, 0, 25);
        buttonPanel.add(stuLogin);
        buttonPanel.add(adminLogin);
        buttonPanel.add(exitProg);

        //Vertical boxlayout add from top to bottom
        JPanel boxPanel = gui.createBoxPanel();
        boxPanel.setLayout(new BoxLayout(boxPanel, BoxLayout.Y_AXIS));
        boxPanel.add(Box.createVerticalStrut(35));
        boxPanel.add(buttonPanel);
        f.add(boxPanel, BorderLayout.CENTER);
        f.setVisible(true);

        /*Invokes login page.
         * If login is successful, goes to respective menu
         */
        ActionListener buttonAction = new ActionListener(){
            public void actionPerformed(ActionEvent e){
                if(e.getSource() == stuLogin){
                    Student u1 = new Student();
                    if(u1.login()){
                        f.dispose();
                        u1.user_menu();
                    }
                }
                else if(e.getSource() == adminLogin){
                    Staff s1 = new Staff();
                    if(s1.login()){
                        f.dispose();
                        staff_menu();
                    }
                }
                else if(e.getSource() == exitProg){
                    f.dispose(); //Terminates frame and program
                    System.exit(0);
                }
            }
        };

        stuLogin.addActionListener(buttonAction);
        adminLogin.addActionListener(buttonAction);
        exitProg.addActionListener(buttonAction);
    }

    static void staff_menu(){
        /*This function is invoked after admin login is successful.
         *  It is the main menu page for admin.
         *  This menu page contains 3 buttons: Process pending request, 
direct to record menu and venue menu.
         */
        final Staff s1 = new Staff();
        final JFrame f = gui.createFrame("Admin Main Menu");
        JLabel l1 = gui.createHeadingLabel("Welcome " + s1.getUsername());
        f.add(l1, BorderLayout.NORTH);

        //Create JButtons for user to click
        final JButton reqProcess = new JButton("Manage Pending Requests");
        final JButton recMenu = new JButton("Manage Records");
        final JButton venMenu = new JButton("Manage Venue");
        final JButton logOut= new JButton("Log out");

        //Create a panel to place buttons vertically
        JPanel buttonPanel = gui.createVertPanel(4, 1, 0, 30);
        buttonPanel.setMaximumSize(new Dimension(300,230));
        buttonPanel.add(reqProcess);
        buttonPanel.add(recMenu);
        buttonPanel.add(venMenu);
        buttonPanel.add(logOut);
        //Vertical boxlayout add from top to bottom
        JPanel boxPanel = gui.createBoxPanel();
        boxPanel.setLayout(new BoxLayout(boxPanel, BoxLayout.Y_AXIS));
        boxPanel.add(Box.createVerticalStrut(20));
        boxPanel.add(buttonPanel);
        f.add(boxPanel, BorderLayout.CENTER);
        f.setVisible(true);

        //When button is clicked, invoke class functions accordingly
        ActionListener buttonAction = new ActionListener(){
            public void actionPerformed(ActionEvent e){
                if(e.getSource() == reqProcess){
                    f.dispose();
                    s1.processRequest();
                }else if(e.getSource() == recMenu){
                    f.dispose();
                    s1.user_menu();
                }
                else if(e.getSource() == venMenu){
                    final Venue v1 = new Venue();
                    f.dispose();
                    v1.venue_menu();
                }
                else if(e.getSource() == logOut){
                    f.dispose(); //Terminates frame and goes back to login
page (main)
                    main(new String[0]);
                }
            }
        };
        reqProcess.addActionListener(buttonAction);
        recMenu.addActionListener(buttonAction);
        venMenu.addActionListener(buttonAction);
        logOut.addActionListener(buttonAction);
    }

    static int countRecord(String fname){
        /*This function calculates the total number of records in a text
file
         * This is unrelated to the record ID.
        */
        int cnt=0;
        try{
            BufferedReader reader = new BufferedReader(new
FileReader(fname));
            while(reader.readLine() != null)
                cnt++;
            reader.close();
        }catch(IOException e){
            //This exception is thrown when error occurs during file
reading
            JOptionPane.showMessageDialog(null, "Failed to count record.",
"Error", JOptionPane.ERROR_MESSAGE);
        }
        return cnt;
    }
    static Vector<String> loadRecord(String fname){
        /*This function loads all records in a text file and return it as a
vector of string */
        Vector<String> allRecord = new Vector<String>();
        try{
            File checkFile = new File(fname);
            /*If the given file name exists, reads the content of the file
             * if !exist, create an empty txt file with the given file
name.
            */
            if(checkFile.exists()){
                FileReader r = new FileReader(fname);
                Scanner s = new Scanner(r);
                while(s.hasNextLine()){
                    allRecord.addElement(s.nextLine());
                }
                r.close();
            }
            else
                checkFile.createNewFile();
        }catch(IOException e){
            //This exception is thrown when error occurs during file
reading or writing
            JOptionPane.showMessageDialog(null, "Failed to load record.",
"Error", JOptionPane.ERROR_MESSAGE);
        }
        return allRecord;
    }
}
```

User.java

```
import java.awt.*;
import java.util.*;
import org.omg.CORBA.Request;
import java.io.*;
import java.nio.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableModel;

/*This class is the abstract superclass of Student and Staff.
 * Refer to its child class for general explanation of the functions
 */
abstract class User{
    protected String[] temp;
    protected Vector<String> allRecord;
    protected int index;
    protected String fieldColumn[] = {"Record ID", "Venue ID", "Start
Date&Time", "End Date&Time", "Description", "Status", "Request Date", "User
ID"};
    protected UVMS uvms = new UVMS();
    protected GUI gui = new GUI(); //Composition: User Class uses/has-a GUI
    private boolean recStatus;
    private Font fieldFont = new Font("Calibri", Font.BOLD, 14);
    private Font recFont = new Font("Calibri", Font.PLAIN, 14);
    private SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy
HH:mm");

    private JFrame viewTable;

    Vector<String> getVector(){
        return allRecord;
    }

    void user_menu(){}

    void addRecord(){
        //Increment the total number of records by 1 as new ID.
        index = uvms.countRecord("record.txt")+1;
        temp[0] = new String(Integer.toString(index));

        //Format and initialize required fields
        temp[5] = "Pending";
        Date date = new Date();
        temp[6] = new String(sdf.format(date));

        Venue v1 = new Venue();
        //Create JComponents for Columns
        JComponent[] ColumnData = new JComponent[8];
        JLabel rid = new JLabel(temp[0],
JLabel.CENTER);                           rid.setFont(recFont);
        final JComboBox vid = new
JComboBox(v1.getAvailableVenue());               vid.setFont(recFont);
        final JTextField startDate = new JTextField("dd-MM-yyyy
hh:mm");        startDate.setFont(recFont);
        final JTextField endDate = new JTextField("dd-MM-yyyy
hh:mm");          endDate.setFont(recFont);
        final JTextField desc = new
JTextField();                                  desc.setFont(recFont);
        JLabel status = new JLabel(temp[5],
JLabel.CENTER);                        status.setFont(recFont);
        JLabel reqDate = new JLabel(temp[6],
JLabel.CENTER);                       reqDate.setFont(recFont);
        JLabel uid = new JLabel(temp[7],
JLabel.CENTER);                           uid.setFont(recFont);
        final JButton displayVenTT = new JButton("Timetable");

        //Create fields as JLabels (left column)
        for(int i=0; i<8; i++){
            ColumnData[i] = new JLabel(fieldColumn[i], JLabel.RIGHT);
            ColumnData[i].setFont(fieldFont);
        }

        //Display New Frame
        final JFrame f = gui.createFrame("Add Record");

        //Add data to their respective cells
        JPanel p = gui.createVertPanel(8, 3, 5, 5);
        p.setMaximumSize(new Dimension(420,230));
        p.add(ColumnData[0]); p.add(rid); p.add(new JLabel());
        p.add(ColumnData[1]); p.add(vid);  p.add(displayVenTT);
        p.add(ColumnData[2]); p.add(startDate); p.add(new JLabel());
        p.add(ColumnData[3]); p.add(endDate); p.add(new JLabel());
        p.add(ColumnData[4]); p.add(desc); p.add(new JLabel());
        p.add(ColumnData[5]); p.add(status); p.add(new JLabel());
        p.add(ColumnData[6]); p.add(reqDate); p.add(new JLabel());
        p.add(ColumnData[7]); p.add(uid); p.add(new JLabel());

        //Create buttons for user to click
        final JButton saveRec = new JButton("Add");
        final JButton cancelAdd = new JButton("Cancel");

        //Create panel to hold buttons
        JPanel buttonPanel = gui.createHoriPanel();
        buttonPanel.add(saveRec);
        buttonPanel.add(cancelAdd);

        //Add panel to box panel and add it to the center of frame
        JPanel boxPanel = gui.createBoxPanel();
        boxPanel.add(p, BorderLayout.CENTER);

        JPanel formatSpaceTop = new JPanel();
        formatSpaceTop.setLayout(new BoxLayout(formatSpaceTop,
BoxLayout.X_AXIS));
        formatSpaceTop.add(Box.createVerticalStrut(20));

        f.add(formatSpaceTop, BorderLayout.NORTH);
        f.add(boxPanel, BorderLayout.CENTER);
        f.add(buttonPanel, BorderLayout.SOUTH);
        f.setVisible(true);

        ActionListener buttonAction = new ActionListener(){
            public void actionPerformed(ActionEvent e){
                //Validate&check clash for dates before saving
                if(e.getSource() == saveRec){
                    //Retrieve new data for editable field (2,3,4)
                    temp[1] = vid.getSelectedItem().toString();
                    temp[2] = startDate.getText();
                    temp[3] = endDate.getText();
                    temp[4] = desc.getText();

                    //Check Clash for Start/End date&time
                    ArrayList<String[]> checkList = new
ArrayList<String[]>();
                    checkList = loadCheckList(temp[1], 0);
                    /* If venue ID is not null, proceed to check dates
                     * If both dates are valid & no clash in time, save
changes; else, display error message
                    */
                    if(!vid.getSelectedItem().toString().equals("null")){
                        if(dateValid(temp[2]) && dateValid(temp[3])){
                            if(!startClash(checkList, temp[2])
&& !endClash(checkList, temp[2], temp[3]) && !overlapClash(checkList,
temp[2], temp[3])){
```

```
                                //If all conditions are met, save new
record in the file (updateRec function)
                                recStatus = true;
                                updateRec(allRecord, index);
                                f.dispose();
                                if(viewTable!=null) viewTable.dispose();
                                user_menu();
                            }
                        }
                    }else{ //Venue ID is null (no venue records available
for users to choose)
                        recStatus = false;
                        JOptionPane.showMessageDialog(null, "No available
venue! Contact admin to add venue.", "Warning",
JOptionPane.WARNING_MESSAGE);
                    }
                }else if(e.getSource() == cancelAdd){ //User cancels the
add process
                    recStatus = false;
                    f.dispose();
                    if(viewTable!=null) viewTable.dispose();
                    user_menu();
                }else if(e.getSource() == displayVenTT){
                    //User wants to display the timetable of a venue
                    if(vid.getSelectedItem().toString().equals("null")){
                        JOptionPane.showMessageDialog(null, "No available
venue.", "Warning", JOptionPane.WARNING_MESSAGE);
                    }else{
                        ArrayList<String[]> checkList = new
ArrayList<String[]>();
                        checkList =
loadCheckList(vid.getSelectedItem().toString(), 0);
                        if(checkList.size()==0)
                            JOptionPane.showMessageDialog(null,
vid.getSelectedItem().toString() + " has no reservation record. All time
slots free.", "Information", JOptionPane.INFORMATION_MESSAGE);
                        else{ //Found reservation records under the venue
chosen. Sort and display venue timetable
                            sortDate(checkList);
                            venueTT(checkList,
vid.getSelectedItem().toString());
                        }
                    }
                }
            }
        };
        saveRec.addActionListener(buttonAction);
        cancelAdd.addActionListener(buttonAction);
        displayVenTT.addActionListener(buttonAction);

        return;
    }

    void editRecord(final int id){
        JComponent[] ColumnData = new JComponent[8];
        //Create JLabels for left column (field)
        for(int i=0; i<8; i++){
            ColumnData[i] = new JLabel(fieldColumn[i], JLabel.RIGHT);
            ColumnData[i].setFont(fieldFont);
        }

        Venue v1 = new Venue();

        /**Create JTextFields for editable information & format the
fields*/
        JLabel rid = new JLabel(temp[0],
JLabel.CENTER);                rid.setFont(recFont);
        final JComboBox vid = new
JComboBox(v1.getAvailableVenue());    vid.setFont(recFont);
        vid.setSelectedItem(temp[1]); //Initial selected item
        final JTextField startDate = new
JTextField(temp[2]);           startDate.setFont(recFont);
        final JTextField endDate = new
JTextField(temp[3]);             endDate.setFont(recFont);
        final JTextField desc = new
JTextField(temp[4]);                desc.setFont(recFont);
        JLabel status = new JLabel(temp[5],
JLabel.CENTER);             status.setFont(recFont);
        JLabel reqDate = new JLabel(temp[6],
JLabel.CENTER);            reqDate.setFont(recFont);
        JLabel uid = new JLabel(temp[7],
JLabel.CENTER);                uid.setFont(recFont);
        final JButton displayVenTT = new JButton("Timetable");

        //Create display frame
        final JFrame f = gui.createFrame("Edit Record");

        JPanel p = gui.createVertPanel(8, 3, 5, 5);

        //Add Components into Columns
        p.add(ColumnData[0]); p.add(rid); p.add(new JLabel());
        p.add(ColumnData[1]); p.add(vid);  p.add(displayVenTT);
        p.add(ColumnData[2]); p.add(startDate); p.add(new JLabel());
        p.add(ColumnData[3]); p.add(endDate); p.add(new JLabel());
        p.add(ColumnData[4]); p.add(desc); p.add(new JLabel());
        p.add(ColumnData[5]); p.add(status); p.add(new JLabel());
        p.add(ColumnData[6]); p.add(reqDate); p.add(new JLabel());
        p.add(ColumnData[7]); p.add(uid); p.add(new JLabel());

        //Create buttons for user to click
        final JButton saveEdit = new JButton("Save");
        final JButton cancelR = new JButton("Cancel Request");
        final JButton goBack = new JButton("Go Back");

        //Create panel to hold buttons
        JPanel buttonPanel = gui.createHoriPanel();
        buttonPanel.add(saveEdit);
        buttonPanel.add(cancelR);
        buttonPanel.add(goBack);

        JPanel boxPanel = gui.createBoxPanel();
        boxPanel.add(p, BorderLayout.CENTER);

        JPanel formatSpaceTop = new JPanel();
        formatSpaceTop.setLayout(new BoxLayout(formatSpaceTop,
BoxLayout.X_AXIS));
        formatSpaceTop.add(Box.createVerticalStrut(20));

        f.add(formatSpaceTop, BorderLayout.NORTH);
        f.add(boxPanel, BorderLayout.CENTER);
        f.add(buttonPanel, BorderLayout.SOUTH);
        f.setVisible(true);

        ActionListener buttonAction = new ActionListener(){
            public void actionPerformed(ActionEvent e){
                if(e.getSource() == saveEdit){
                    //Retrieve new data for editable field
                    temp[1] = vid.getSelectedItem().toString();
                    temp[2] = startDate.getText();
                    temp[3] = endDate.getText();
                    temp[4] = desc.getText();
                    temp[6] = sdf.format(new Date()); //Update request date

                    //Check Clash for Start/End date&time
                    ArrayList<String[]> checkList = new
ArrayList<String[]>();
                    checkList = loadCheckList(temp[1], id);
                    /* If venue ID is not null, proceed to check dates
                     * If both dates are valid & no clash in time, save
changes; else, display error message
                    */
                    if(!vid.getSelectedItem().toString().equals("null")){
                        if(dateValid(temp[2]) && dateValid(temp[3])){
                            if(!startClash(checkList, temp[2])
&& !endClash(checkList, temp[2], temp[3]) && !overlapClash(checkList,
temp[2], temp[3])){
                                //If all conditions are met, save the
update in the file (updateRec function)
                                recStatus = true;
                                updateRec(allRecord, index);
                                f.dispose();
                                if(viewTable!=null) viewTable.dispose();
                                editRecord(id);
                            }
                        }
                    }else{
                        recStatus = false;
                        JOptionPane.showMessageDialog(null, "No available
venue! Contact admin to add venue.", "Warning",
JOptionPane.WARNING_MESSAGE);
                    }
                }else if(e.getSource() == cancelR){
                    deleteRecord(id);
                    f.dispose();
                    if(viewTable!=null) viewTable.dispose();
                    user_menu();
                }else if(e.getSource() == goBack){
                    f.dispose();
                    if(viewTable!=null) viewTable.dispose();
                    user_menu();
                }else if(e.getSource() == displayVenTT){
                    //User wants to display timetable for the selected 
venue
                    if(vid.getSelectedItem().toString().equals("null")){
                        //Venue ID is null. No available venue for user to 
choose.
                        JOptionPane.showMessageDialog(null, "No available 
venue.", "Warning", JOptionPane.WARNING_MESSAGE);
                    }else{
                        ArrayList<String[]> checkList = new 
ArrayList<String[]>();
                        checkList = 
loadCheckList(vid.getSelectedItem().toString(), 0);
                        if(checkList.size()==0) //Venue ID is valid, Venue 
exists and all time slots are available (no reservation under the venue ID)
                            JOptionPane.showMessageDialog(null, 
vid.getSelectedItem().toString() + " has no reservation record. All time 
slots free.", "Information", JOptionPane.INFORMATION_MESSAGE);
                        else{ //Found reservation records under the venue 
chosen. Sort and display venue timetable
                            sortDate(checkList);
                            venueTT(checkList, 
vid.getSelectedItem().toString());
                        }
                    }
                }
            }
        };
        saveEdit.addActionListener(buttonAction);
        cancelR.addActionListener(buttonAction);
        goBack.addActionListener(buttonAction);
        displayVenTT.addActionListener(buttonAction);
    }

    String[][] viewRecord(Vector<String> records){
        //Each record uses 1 row.
        //Each row has 8 columns/fields
        String[][] data = new String[records.size()][8];
        for(int i=0; i<records.size(); i++){
            temp = new String[0];
            temp = records.get(i).split("\t");
            for(int j=0; j<8; j++){
                data[i][j] = temp[j]; //Insert data into respective row and 
column
            }
        }
        return data;
    }

    void deleteRecord(int input){
        int ans;
        String str = new String();

        //Ask user for delete confirmation
        ans = JOptionPane.showConfirmDialog(null, "Do you want to cancel 
this reservation?", "Confirmation", JOptionPane.YES_NO_OPTION, 
JOptionPane.INFORMATION_MESSAGE);
```

```
        //User selects NO option or closes the dialog
        if(ans == JOptionPane.NO_OPTION || ans == 
JOptionPane.CLOSED_OPTION){
        }else{
            temp = new String[0];
            temp = allRecord.get(input-1).split("\t");
            temp[5] = "Cancelled"; //switch record status to cancelled

            //Update vector
            for(int i=0; i<temp.length; i++){
                str += temp[i] + "\t";
            }
            allRecord.set(input-1, str);

            //Write vector into file
            try{
                writeFile("record.txt", allRecord);
                JOptionPane.showMessageDialog(null, "Reservation is 
cancelled sucessfully!", "Success", JOptionPane.PLAIN_MESSAGE);
            }catch(IOException ioe){
                JOptionPane.showMessageDialog(null, "Error updating file.", 
"Error", JOptionPane.ERROR_MESSAGE);

            }
        }
    }

    boolean searchRecord(int input, Vector<String> records){
        //Returns true if input equals to the record id
        boolean found = false;
        if(temp[0].equals(Integer.toString(input)))
            found = true;
        return found;
    }

    public void updateRec(Vector<String> records, int index){
        /*This function saves all changes made to the records (new record 
and edited record) */
        String str = new String();
        String msg;
        if(recStatus){ //make sure changes need to be made
            for(int i=0; i<temp.length; i++){
                if(temp[i].length()==0) temp[i]="None";
                str += temp[i] + "\t";
            }
            try{ //Update vector
                records.set(index-1, str);
                msg = new String("Record has been updated successfully!");
            }catch(ArrayIndexOutOfBoundsException arrE){
                //OutOfBounds: index is a new record
                records.addElement(str);
                msg = new String("Record has been added successfully!");
            }

            try{ //write vector into file
                writeFile("record.txt", records);
                JOptionPane.showMessageDialog(null, msg, "Success", 
JOptionPane.PLAIN_MESSAGE);
            }catch(IOException ioe){
                JOptionPane.showMessageDialog(null, "Operation failed!", 
"Error", JOptionPane.ERROR_MESSAGE);
            }
            allRecord = records;
        }
    }

    void writeFile(String fname, Vector<String> allRecord)throws 
IOException{
        /*This function updates the Vector into the given file name*/
        FileWriter writeFile = new FileWriter(fname, false);
        String[] temp = new String[0];
        for(int i=0; i<allRecord.size(); i++){
            temp = allRecord.get(i).split("\t");
            for(int j=0; j<temp.length; j++)
                writeFile.write(temp[j] + "\t");
            writeFile.write("\r\n");
            //write in new line, prepare for next record
        }
        writeFile.close();
    }

    boolean dateValid(String d){
        /*This function checks the given date is in correct format and is 
larger than the current date&time */
        try {
            Date currDate = new Date();
            if(sdf.parse(d).compareTo(currDate)<0){
                JOptionPane.showMessageDialog(null, "Input date & time has 
already passed!", "Warning", JOptionPane.WARNING_MESSAGE);
                return false;
            }
        } catch (ParseException e) {
            JOptionPane.showMessageDialog(null, "Invalid date input! ", 
"Error", JOptionPane.ERROR_MESSAGE);
            return false;
        }
        return true;
    }

    Date convertDate(String s){
        Date date = new Date();
        try {
            date = sdf.parse(s);
        } catch (ParseException e) {
            JOptionPane.showMessageDialog(null, "Invalid date input! ", 
"Error", JOptionPane.ERROR_MESSAGE);

        }
        return date;
    }

    void sortDate(ArrayList<String[]> list){
        /*This function sorts the list of dates given in ascending order */
        Collections.sort(list, new Comparator<String[]>() {
            @Override
            public int compare(String[] a, String[] b) {
                return a[2].compareTo(b[2]);
            }
        });
    }

    ArrayList<String[]> loadCheckList(String venueId, int id){
        /*This function loads all approved records under the given venue ID 
into a check list
         * Records before current date&time & in pending/denied/cancelled 
state will not be loaded
         * The list is then returned as a list of entries that needs to be 
checked for time clash
         */

        allRecord = uvms.loadRecord("record.txt");
        String[] temp2;
        ArrayList<String[]> list = new ArrayList<String[]>();

        for(int i=0; i<allRecord.size(); i++){
            temp2 = new String[0]; //clear content of array
            temp2 = allRecord.get(i).split("\t");
            try{
                Date checkEndDate = sdf.parse(temp2[3]);
                Date currDate = new Date();
                //only check for approved request that havent 
complete/finish, past request/before curr date no need check
                if(temp2[1].equals(venueId) && temp2[5].equals("Approved") 
&& currDate.compareTo(checkEndDate) < 0){
                    if(!temp2[0].equals(Integer.toString(id)))
                        list.add(temp2); //record that need to check 
clashing
                }
            }catch(ParseException e){
                JOptionPane.showMessageDialog(null, "Invalid data! ", 
"Error", JOptionPane.ERROR_MESSAGE);
            }
        }
        return list;
    }

    void venueTT(ArrayList<String[]> list, String venueId){
        /*This function displays a series of valid records under the same 
venue ID in ascending order
         */
        //Load the list of date&time and put them in their respective array 
element
        String data[][] = new String[list.size()][3];
        for(int i=0; i<list.size(); i++){
            for(int j=0; j<3; j++){
                data[i][j] = list.get(i)[j+1];
            }
        }

        //Create a table using the loaded data
        String[] venueColumn = {"Venue ID", "Start Date & Time", "End Date
& Time"};
        JTable table = new JTable(data, venueColumn);
        JScrollPane sp = new JScrollPane(table);
        table.setDefaultEditor(Object.class, null);
        table.setAutoCreateRowSorter(true);

        //Align all field at center
        DefaultTableCellRenderer centerRender = new
DefaultTableCellRenderer();
        centerRender.setHorizontalAlignment(SwingConstants.CENTER);
        TableModel tableModel = table.getModel();
        for (int i=0; i<tableModel.getColumnCount(); i++)
            table.getColumnModel().getColumn(i).setCellRenderer(centerRende
r);

        //Create display frame for the page
        if(viewTable != null) viewTable.dispose();
        viewTable = new JFrame("Reservation Timetable of " + venueId);
        viewTable.setSize(1000, 500);
        viewTable.setLayout(new BorderLayout());
        viewTable.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE
);
        viewTable.setLocationRelativeTo(null);
        viewTable.add(sp, BorderLayout.CENTER);
        viewTable.setVisible(true);
    }

    boolean startClash(ArrayList<String[]> list, String s){
        /*This function checks if clashing occurs between 2 given start
date&time*/
        Date start = convertDate(s);
        for(int i=0; i<list.size();
++i){ //sdf.parse(list.get(i)[2]).compareTo(currDate)<0
            if(start.compareTo(convertDate(list.get(i)[2]))>=0 &&
start.compareTo(convertDate(list.get(i)[3]))<0){
                JOptionPane.showMessageDialog(null, "Clashed with " +
list.get(i)[2] + " - " + list.get(i)[3], "Alert",
JOptionPane.WARNING_MESSAGE);
                return true;
            }
        }
        return false;
    }

    boolean endClash(ArrayList<String[]> list, String s, String e){
        /*This function checks if clashing occurs between 2 given end
date&time*/
        Date start = convertDate(s);
        Date end = convertDate(e);
        if(end.compareTo(start)<0){
            JOptionPane.showMessageDialog(null, "End Date & Time is before
Start Date & Time!", "Alert", JOptionPane.WARNING_MESSAGE);
            return true;
        } else if(end.compareTo(start)==0){
            JOptionPane.showMessageDialog(null, "End Date & Time is same as
Start Date & Time!", "Alert", JOptionPane.WARNING_MESSAGE);
```

```
            return true;
        } else{
            for(int i=0; i<list.size(); ++i){
                if(end.compareTo(convertDate(list.get(i)[2]))>0 &&
end.compareTo(convertDate(list.get(i)[3]))<=0){
                    JOptionPane.showMessageDialog(null, "Clashed with " +
list.get(i)[2] + " - " + list.get(i)[3], "Alert",
JOptionPane.WARNING_MESSAGE);
                    return true;
                }
            }
        }
        return false;
    }

    boolean overlapClash(ArrayList<String[]> list, String s, String e){
        /*This function checks if 2 given date&time overlaps each other */
        Date start = convertDate(s);
        Date end = convertDate(e);
        for(int i=0; i<list.size(); ++i){
            if(start.compareTo(convertDate(list.get(i)[2]))<0 &&
start.compareTo(convertDate(list.get(i)[3]))<0 &&
               end.compareTo(convertDate(list.get(i)[2]))>0 &&
end.compareTo(convertDate(list.get(i)[3]))>0){
                    JOptionPane.showMessageDialog(null, "Clash with " +
list.get(i)[2] + " - " + list.get(i)[3], "Alert",
JOptionPane.WARNING_MESSAGE);
                    return true;
            }
        }
        return false;
    }
}
```

Student.java

```
import java.awt.*;
import java.util.*;
import javax.lang.model.util.ElementScanner6;
import org.omg.CORBA.Request;
import java.io.*;
import java.nio.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.JTable;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableModel;

/*This class refers to any user that is non-admin (Student, Lecturers)
 * In the explanation, non-admin will be referred as a student for better
clarification.
 * Student Class inherits User class: Student is a User.
 */
public class Student extends User{
    private String pwd;
    private String uid;

    Student(){} //constructor

    boolean login(){
        /*Student login function
         * This function returns true indicating successful login. (uid and 
password matches)
         */
        boolean status = false;
        int ans;
        Vector<String> userRecord = uvms.loadRecord("login.txt");

        while(true){
            while(true){
                try{
                    //Prompt student to enter a user ID and validates the 
input. UID cannot be null or admin.
                    uid = JOptionPane.showInputDialog(null, "Enter 
Username", "");
                    if(uid == null)
                        return status;
                    else if(uid.equals(""))
                        throw new NullPointerException();
                    else if(uid.toLowerCase().equals("admin"))
                        JOptionPane.showMessageDialog(null, "Cannot login 
as admin!", "Warning", JOptionPane.WARNING_MESSAGE);
                    else
                        break;
                }catch(NullPointerException npe){
                    JOptionPane.showMessageDialog(null, "You must enter an 
input!", "Error", JOptionPane.ERROR_MESSAGE);
                }
            }

            //Checks if user exists within login.txt file.
            for(int i=0; i<userRecord.size(); i++){
                temp = new String[0];
                temp = userRecord.get(i).split("\t");
                if(temp[0].equals(uid)){
                    while(true){
                        //If UID exists, keep looping until there is a 
proper input or dialog is cancelled/closed
                        try{
                            pwd = JOptionPane.showInputDialog(null, "Enter 
Password for user " + uid, ""); //Display message and initial selection 
value
                            if(pwd == null) //Input dialog cancelled/closed
                                return status;
                            else if(pwd.equals("")) //input is empty
                                throw new NullPointerException();
                            else if(pwd.equals(temp[1])){ //only set login 
status as true if password matches
                                status = true;
                                break;
                            }
                            else
                                JOptionPane.showMessageDialog(null, "Wrong 
login credentials!", "Information", JOptionPane.INFORMATION_MESSAGE);
                        }catch(NullPointerException npe){
                            JOptionPane.showMessageDialog(null, "You must 
enter an input!", "Error", JOptionPane.ERROR_MESSAGE);
                        }
                    }
                    break;
                }
            }
            if(temp[0].equals(uid)) //if temp[0] = uid, means there is a 
record that matches the password. (because multiple accounts with same 
password may occur)
                break;

            //if there is no matching credentials, ask if the student wants 
to create a new account
            ans = JOptionPane.showConfirmDialog(null, "User does not exist! 
Do you want to create new user?", "Confirm Registration", 
JOptionPane.YES_NO_OPTION, JOptionPane.INFORMATION_MESSAGE);
            if(ans == JOptionPane.YES_OPTION){ //student chooses yes 
option. invoke register user function
                registerUser(userRecord);
                break;
            }
        }
        return status;
    }

    void registerUser(Vector<String> userRecord){
        /*This function is invoked from student login function. Creates a 
new user and write it into login.txt file */
        boolean exist = false;
        while(true){
            try{
                //Ask for new username input
                uid = JOptionPane.showInputDialog(null, "Register new 
username", "");
                if(uid == null) //Input Dialog is cancelled/closed
                    return;
                else if(uid.toLowerCase().equals("admin")) //New username 
cannot be admin
                    JOptionPane.showMessageDialog(null, "Username cannot be 
admin!", "Warning", JOptionPane.WARNING_MESSAGE);
                else if(uid.equals("")) //input is empty
                    throw new NullPointerException();
                else{
                    //check if user exists
                    for(int i=0; i<userRecord.size(); i++){
                        temp = new String[0];
                        temp = userRecord.get(i).split("\t");
                        if(uid.equals(temp[0])){
                            exist = true;
                            break;
                        }
                    }
                    if(exist){ //continues to loop for new username input
if current username is already taken
                        JOptionPane.showMessageDialog(null, "User already
exists!", "Alert", JOptionPane.WARNING_MESSAGE);
                    }else{//if username is not taken, ask for new password
                        try{
                            pwd = JOptionPane.showInputDialog(null, "Enter
password for new user " + uid + ": ", "Register New User",
JOptionPane.INFORMATION_MESSAGE);
                            if(pwd == null) //Input dialog is
cancelled/closed
                                return;
                            //Save uid and password into file
                            String str = new String(uid + "\t" + pwd);
                            userRecord.addElement(str);
                            writeFile("login.txt", userRecord);
                            JOptionPane.showMessageDialog(null, "User
created successfully!", "Success", JOptionPane.PLAIN_MESSAGE);
                            break;
                        }catch(IOException e){
                            JOptionPane.showMessageDialog(null, "New user
creation failed.", "Warning", JOptionPane.WARNING_MESSAGE);
                        }
                    }
                }
            }catch(NullPointerException npe){
                JOptionPane.showMessageDialog(null, "You must enter an
input!", "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    String getUsername(){
        return uid;
    }

    void user_menu(){
        /*This function is invoked after student login is successful.
         *  It is the main menu page for students
         */
        final String fname = new String("record.txt");

        /*Create a display frame for the function */
        final JFrame f = gui.createViewPage("Student Dashboard");
        JLabel l1 = gui.createHeadingLabel("Welcome " + getUsername());

        //Create JButtons for user to click
        final JButton addReq = new JButton("New reservation request");
        final JButton logOut = new JButton("Log out");

        //Create a panel to place buttons horizontally
        JPanel buttonPanel = gui.createHoriPanel();
        buttonPanel.add(addReq);
        buttonPanel.add(logOut);

        //A Panel to hold items on north
        JPanel NorthPanel = gui.createVertPanel(2,1,0,0);
        NorthPanel.add(l1);
        NorthPanel.add(buttonPanel);

        Vector<String> records = uvms.loadRecord(fname);
        String data[][] = viewRecord(records);

        //Create a table to place the data.
        final JTable table = new JTable(data, fieldColumn);
        JScrollPane sp = new JScrollPane(table);
        table.setDefaultEditor(Object.class, null);
        table.setAutoCreateRowSorter(true);

        //Set table alignment to center
        DefaultTableCellRenderer centerRender = new 
DefaultTableCellRenderer();
        centerRender.setHorizontalAlignment(SwingConstants.CENTER);
        TableModel tableModel = table.getModel();
        for (int i=0; i<tableModel.getColumnCount(); i++)
            table.getColumnModel().getColumn(i).setCellRenderer(centerRende
r);

        f.add(sp, BorderLayout.CENTER);
        f.add(NorthPanel, BorderLayout.NORTH);

        //Display alert message at the bottom
        JPanel msgPanel = gui.createHoriPanel();
        JLabel instruction = gui.BottomMessage();
        msgPanel.add(instruction);
        f.add(msgPanel, BorderLayout.SOUTH);
        f.setVisible(true);

        //When button is clicked, invoke class functions accordingly
        ActionListener buttonAction = new ActionListener(){
            public void actionPerformed(ActionEvent e){
                if(e.getSource() == addReq){
                    f.dispose();
                    allRecord = uvms.loadRecord(fname);
                    addRecord();
                }
                else if(e.getSource() == logOut){
                    //Terminates frame and returns to login page (main)
                    f.dispose();
                    uvms.main(new String[0]);
                }
            }
        };
        addReq.addActionListener(buttonAction);
        logOut.addActionListener(buttonAction);

        table.getSelectionModel().addListSelectionListener(new 
ListSelectionListener(){
            public void valueChanged(ListSelectionEvent e){
                if(e.getValueIsAdjusting()){
                    //System.out.println(table.getValueAt(table.getSelected
Row(), 0).toString());
```

```
                    int id =
Integer.parseInt(table.getValueAt(table.getSelectedRow(), 0).toString());
                    f.dispose();
                    allRecord = uvms.loadRecord(fname);
                    editRecord(id);
                }
            }
        });

    }

    void addRecord(){
        /*This function adds a new reservation REQUEST in the system.
         * Status of all new requests added by users will be set as
"Pending"
         * Students are required to wait until admin approves the request,
then the request will be considered as a valid record.
        */
        //Initialize temp[] and set uid for display later
        super.temp = new String[8];
        super.temp[7] = new String(uid);
        //Invoke super class method
        super.addRecord();
        return;
    }

    void editRecord(int id){
        /*This function allows the student to edit any reservation request
that belongs to their user ID
         * Status of all edited request will be set to "Pending",
considered as a new request
         * Editable fields are Venue ID, Start/End Date&Time, Description.
         */

        //Initialize temp[] and retrieve record
        super.temp = new String[0];
        super.temp = allRecord.get(id-1).split("\t");
        super.index = id; //Index of the record within the vector is saved
        super.editRecord(id);
        super.temp[5] = "Pending";
    }

    String[][] viewRecord(Vector<String> records){
        /*This function lets student to view all reservation requests
available UNDER THEIR UID
        */
        Vector<String> tempVector = new Vector<String>(); //initialize
before use
        for(int i=0; i<records.size(); i++){
            temp = new String[0]; //clear content of array
            temp = records.get(i).split("\t");
            //Filter out records that belongs to the student to a vector
            if(temp[7].equals(uid)){
                tempVector.addElement(records.get(i));
            }
        }
        //Pass the filtered results to view
        return super.viewRecord(tempVector);
    }

    void deleteRecord(int input){
        /*This function allows students to cancel a reservation record
(Approved/Pending requests)
         * The function does not delete the record straightaway as the
students can refer/edit their request in the future
        */

        temp = new String[0];
        temp = allRecord.get(input-1).split("\t");
        if(temp[5].equals("Approved") || temp[5].equals("Pending")) //Can
only cancel a request if record is approved or pending
            super.deleteRecord(input);
        else
            JOptionPane.showMessageDialog(null, "Reservation is already in
cancelled or denied state!", "Information",
JOptionPane.INFORMATION_MESSAGE);
    }

    boolean searchRecord(int input, Vector<String> records){
        /*This function checks if the input record ID exists and returns as
boolean
         * Students may access to records under their UID only
         */
        boolean found = false;

        //Scan if the record id matches input (superclass method)
        for(int i=0; i<records.size(); i++){
            super.temp = new String[0];
            super.temp = records.get(i).split("\t");
            if(super.temp[7].equals(uid) && super.searchRecord(input,
records)){ //return true if record id matches input & record belongs to
user
                    found = true;
                    break;
            }
        }
        return found;
    }
}
```

Staff.java

```
import java.awt.*;
import java.util.*;
import org.omg.CORBA.Request;
import java.io.*;
import java.nio.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.JTable;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableModel;

/*This class refers to Admin.
 * Staff class inherits User class: Admin is a User.
 */
public class Staff extends User{
    private String pwd; //username = user id
    private String uid = new String("Admin");

    Staff(){} //constructor

    boolean login(){
        /*Staff login function
         * This function returns true indicating successfully login
(password is correct)
         */
        boolean status = false;
        Vector<String> userRecord = uvms.loadRecord("login.txt");

        while(true){ //loop until password is correct
            try{
                pwd = JOptionPane.showInputDialog(null, "Enter Admin
Password", "");
                if(pwd == null) //Input dialog cancelled/closed
                    return status;
                else if(pwd.equals("")) //input is empty
                    throw new NullPointerException();
                else{
                    temp = new String[0];
                    temp = userRecord.get(0).split("\t"); //admin will
always be 1st record (index 0)
                    if(pwd.equals(temp[1])){
                        status = true;
                        break; //return true and break from loop if
password is correct
                    }
                }
                JOptionPane.showMessageDialog(null, "Wrong password!",
"Information", JOptionPane.INFORMATION_MESSAGE);
            }catch(NullPointerException npe){
                JOptionPane.showMessageDialog(null, "You must enter an
input!", "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
        return status;
    }

    String getUsername(){
        return uid;
    }

    void user_menu(){
        /*This function is invoked after staff login is successful.
         *  It is the main menu page for admin
         */
        final String fname = new String("record.txt");
        /*Create a display frame for the function */
        final JFrame f = gui.createViewPage("Admin Dashboard");
        JLabel l1 = gui.createHeadingLabel("Welcome " + getUsername());

        //Create JButtons for user to click
        final JButton addRec = new JButton("Add new record");
        final JButton exitPage = new JButton("Return to main menu");

        //Create a panel to place buttons horizontally
        JPanel buttonPanel = gui.createHoriPanel();
        buttonPanel.add(addRec);
        buttonPanel.add(exitPage);

        //A Panel to hold items on north
        JPanel NorthPanel = gui.createVertPanel(2,1,0,0);
        NorthPanel.add(l1);
        NorthPanel.add(buttonPanel);

        Vector<String> records = uvms.loadRecord(fname);
        String data[][] = viewRecord(records);

        //Create a table to place the data.
        final JTable table = new JTable(data, fieldColumn);
        JScrollPane sp = new JScrollPane(table);
        table.setDefaultEditor(Object.class, null);
        table.setAutoCreateRowSorter(true);

        //Set table alignment to center
        DefaultTableCellRenderer centerRender = new
DefaultTableCellRenderer();
        centerRender.setHorizontalAlignment(SwingConstants.CENTER);
        TableModel tableModel = table.getModel();
        for (int i=0; i<tableModel.getColumnCount(); i++)
            table.getColumnModel().getColumn(i).setCellRenderer(centerRende
r);

        f.add(sp, BorderLayout.CENTER);
        f.add(NorthPanel, BorderLayout.NORTH);

        //Display alert message at the bottom
        JPanel msgPanel = gui.createHoriPanel();
        JLabel instruction = gui.BottomMessage();
        msgPanel.add(instruction);
        f.add(msgPanel, BorderLayout.SOUTH);
        f.setVisible(true);

        //When button is clicked, invoke class functions accordingly
        ActionListener buttonAction = new ActionListener(){
            public void actionPerformed(ActionEvent e){
                if(e.getSource() == addRec){
                    f.dispose();
                    allRecord = uvms.loadRecord(fname);
                    addRecord();
                }
                else if(e.getSource() == exitPage){
                    //Terminates frame and returns to login page (main)
                    f.dispose();
                    uvms.staff_menu();
                }
            }
        };
        addRec.addActionListener(buttonAction);
        exitPage.addActionListener(buttonAction);

        table.getSelectionModel().addListSelectionListener(new
ListSelectionListener(){
            public void valueChanged(ListSelectionEvent e){
                if(e.getValueIsAdjusting()){
                    int id =
Integer.parseInt(table.getValueAt(table.getSelectedRow(), 0).toString());
                    f.dispose();
                    allRecord = uvms.loadRecord(fname);
                    editRecord(id);
                }
            }
        });

    }

    void addRecord(){
        /*This function adds a new reservation RECORD in the system.
         * Status of all new records added by admin will be set as
"Approved"
         */
        //Initialize temp[] and set uid for display later
        super.temp = new String[8];
        super.temp[7] = new String(uid);
        //Invoke superclass method
        super.addRecord();
        super.temp[5] = new String("Approved");
        return;
    }

    void editRecord(int id){
        /*This function allows the admin to edit a reservation record.
         * Editable fields are Venue ID, Start/End Date&Time, Description.
         * Status will be automatically set as "Approved" for editted
records
        */

        //Initialize temp[] and retrieve records.
        super.temp = new String[0];
        super.temp = allRecord.get(id-1).split("\t");
        super.index = id; //Index of the record within the vector is saved
        super.editRecord(id);
        super.temp[5] = "Approved";
    }

    String[][] viewRecord(Vector<String> records){
        /*This function lets admin view all reservation records available
in the system.
         * No condition needs to be met for admin. (Default superclass
function)
         */
        return super.viewRecord(records);
    }

    void deleteRecord(int input){
        /*This function allows admin to switch a record status to cancel.
         * The function does not delete the record straightaway as the
students need to see that the record is cancelled (not disappear)
         */

        temp = new String[0];
        temp = allRecord.get(input-1).split("\t");
        if(temp[5].equals("Approved") || temp[5].equals("Pending")) //Can
only cancel a record if status is approved or pending
            super.deleteRecord(input);
        else
            JOptionPane.showMessageDialog(null, "Reservation is already in
cancelled or denied state!", "Information",
JOptionPane.INFORMATION_MESSAGE);
    }

    boolean searchRecord(int input, Vector<String> records){
        /*This function checks if the input record ID exists and returns it
as a boolean
         * Admin may search/access all record without limitations
        */
        boolean found = false;
        //Thus, admin uses default search function (superclass method)
        for(int i=0; i<records.size(); i++){
            super.temp = new String[0];
            super.temp = records.get(i).split("\t");
            if(super.searchRecord(input, records)){
                    found = true;
                    break;
            }
        }
        return found;
    }

    void processRequest(){
        /*This function allows admin to approve/deny pending requests made
by students */
        allRecord = uvms.loadRecord("record.txt");
        Vector<String> tempVector = new Vector<String>();

        //Filter & save all pending request in a temporary vector
        for(int i=0; i<allRecord.size(); i++){
            temp = new String[0];
            temp = allRecord.get(i).split("\t");
            if(temp[5].equals("Pending"))
                tempVector.addElement(allRecord.get(i));
        }

        //Lets admin view all filtered records (pending requests)
        if(tempVector.size()==0){
            JOptionPane.showMessageDialog(null, "No pending requests.",
"Information", JOptionPane.INFORMATION_MESSAGE);
            uvms.staff_menu();
            return;
        }

        final JFrame f = gui.createViewPage("Manage Pending Requests");
        JLabel l1 = gui.createHeadingLabel("Welcome " + getUsername());

        //Create JButtons for user to click
        final JButton exitPage = new JButton("Return to main menu");
        JPanel buttonPanel = gui.createHoriPanel();
        buttonPanel.add(exitPage);

        //Vertical boxlayout add from top to bottom
        JPanel NorthPanel = gui.createVertPanel(2,1,0,0);
        NorthPanel.add(l1);
        NorthPanel.add(buttonPanel);

        String data[][] = viewRecord(tempVector);
        final JTable table = new JTable(data, fieldColumn);
        JScrollPane sp = new JScrollPane(table);
        table.setDefaultEditor(Object.class, null);

        //Set table alignment to center
        DefaultTableCellRenderer centerRender = new
DefaultTableCellRenderer();
        centerRender.setHorizontalAlignment(SwingConstants.CENTER);
        TableModel tableModel = table.getModel();
        for (int i=0; i<tableModel.getColumnCount(); i++)
            table.getColumnModel().getColumn(i).setCellRenderer(centerRende
r);
        f.add(sp, BorderLayout.CENTER);
        f.add(NorthPanel, BorderLayout.NORTH);

        //Display alert message at the bottom
        JPanel msgPanel = gui.createHoriPanel();
        JLabel instruction = gui.BottomMessage();
        msgPanel.add(instruction);
        f.add(msgPanel, BorderLayout.SOUTH);
        f.setVisible(true);

        //When button is clicked, invoke class functions accordingly
        ActionListener buttonAction = new ActionListener(){
            public void actionPerformed(ActionEvent e){
                if(e.getSource() == exitPage){
                    //Terminates frame and returns to login page (main)
                    f.dispose();
                    uvms.staff_menu();
                }
            }
        };
        exitPage.addActionListener(buttonAction);

        //Invoke approveReq() if user clicks on a record
        table.getSelectionModel().addListSelectionListener(new
ListSelectionListener(){
            public void valueChanged(ListSelectionEvent e){
                if(e.getValueIsAdjusting()){
                    int id = 
Integer.parseInt(table.getValueAt(table.getSelectedRow(), 0).toString());
                    approveReq(id);
                    f.dispose();
                    processRequest();
                }
            }
        });
    }

    void approveReq(int id){
        int ans;
        String str = new String();

        ans = JOptionPane.showConfirmDialog(null, "Do you approve this 
request?", "Confirmation", JOptionPane.YES_NO_OPTION, 
JOptionPane.INFORMATION_MESSAGE);
        temp = new String[0];
        temp = allRecord.get(id-1).split("\t");

        if(ans == JOptionPane.CLOSED_OPTION)
            return; //Dialog is closed
        else if(ans == JOptionPane.YES_OPTION){
            //Run check clash if wants to approve record. No changes if 
date clashed
            ArrayList<String[]> checkList = new ArrayList<String[]>();
            checkList = loadCheckList(temp[1], 0);
            if(dateValid(temp[2]) && dateValid(temp[3])){
                if(startClash(checkList, temp[2]) || endClash(checkList, 
temp[2], temp[3]) || overlapClash(checkList, temp[2], temp[3])){
                    return;
                } else temp[5] = "Approved";
            }
        }
        else //NO option
            temp[5] = "Denied";

        //Update Vector
        for(int i=0; i<temp.length; i++)
            str += temp[i] + "\t";
        allRecord.set(id-1, str);
        //Update File
        try{
            writeFile("record.txt", allRecord);
            JOptionPane.showMessageDialog(null, "Record has been updated 
successfully!", "Success", JOptionPane.PLAIN_MESSAGE);
        }catch(IOException ioe){
            JOptionPane.showMessageDialog(null, "Record update failed.", 
"Error", JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

Venue.java

```
import java.awt.*;
import java.util.*; //java.util.Scanner = take input
import org.omg.CORBA.Request;
import java.io.*;
import java.nio.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.awt.event.*;
import javax.lang.model.util.ElementScanner6;
import javax.swing.*;
import javax.swing.plaf.BorderUIResource;
import javax.swing.event.*;
import javax.swing.JTable;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableModel;

/*This class allows admin to manage and manipulate all available venue 
information */
public class Venue{
    protected String[] temp;
    protected Vector<String> allRecord;
    private boolean recStatus;
    private Font fieldFont = new Font("Ariel", Font.BOLD, 14);
    private Font recFont = new Font("Calibri", Font.PLAIN, 14);
    private String[] fieldOption = {"Venue ID", "Venue Name", "Venue Type", 
"Description"};
    private String[] venueType = {"Tutorial Room", "Lecture Room", "Lecture 
Hall", "Laboratory"};
    private GUI gui = new GUI();
    private UVMS uvms = new UVMS();

    Vector<String> getVector(){
        return allRecord;
    }

    Vector<String> getAvailableVenue(){
        /*This function loads all available Venue ID and returns it as a 
vector */
        Vector<String> venueList = new Vector<String>();

        //Read each record and store the Venue ID in venue List
        Vector<String> temp2 = uvms.loadRecord("venue.txt");

        //Indicate there is no available venue if size is 0
        if(temp2.size()==0)
            venueList.addElement("null");
        else{
            //Push only venue ID into the vector
            for(int i=0; i<temp2.size(); i++){
                String[] strTemp = new String[0];
                strTemp = temp2.get(i).split("\t");
                venueList.addElement(strTemp[1]);
            }
        }

        return venueList;
    }

    void venue_menu(){
        /*
         *  It is the main menu page for venue
         */
        final String fname = new String("venue.txt");
        final Venue v1 = new Venue();

        /*Create a display frame for the function */
        final JFrame f = gui.createViewPage("Admin Dashboard");
        JLabel l1 = gui.createHeadingLabel("Manage Venue");
        f.add(l1, BorderLayout.NORTH);

        //Create JButtons for user to click
        final JButton addVen = new JButton("Add new venue");
        final JButton exitPage = new JButton("Return to main menu");

        //Create a panel to place buttons horizontally
        JPanel buttonPanel = gui.createHoriPanel();
        buttonPanel.add(addVen);
        buttonPanel.add(exitPage);

        //A Panel to hold items on north
        JPanel NorthPanel = gui.createVertPanel(2,1,0,0);
        NorthPanel.add(l1);
        NorthPanel.add(buttonPanel);

        Vector<String> records = uvms.loadRecord(fname);
        String data[][] = viewRecord(records);

        //Create a table to place the data
        String column[] = {"Venue ID", "Name", "Venue Type",
"Description"};
        final JTable table = new JTable(data, column);
        JScrollPane sp = new JScrollPane(table);
        table.setDefaultEditor(Object.class, null);
        table.setAutoCreateRowSorter(true);

        //Set table alignment to center
        DefaultTableCellRenderer centerRender = new
DefaultTableCellRenderer();
        centerRender.setHorizontalAlignment(SwingConstants.CENTER);
        TableModel tableModel = table.getModel();
        for (int i=0; i<tableModel.getColumnCount(); i++)
            table.getColumnModel().getColumn(i).setCellRenderer(centerRende
r);

        f.add(sp, BorderLayout.CENTER);
        f.add(NorthPanel, BorderLayout.NORTH);

        //Display alert message at the bottom
        JPanel msgPanel = gui.createHoriPanel();
        JLabel instruction = gui.BottomMessage();
        msgPanel.add(instruction);
        f.add(msgPanel, BorderLayout.SOUTH);
        f.setVisible(true);

        //When button is clicked, invoke class functions accordingly
        ActionListener buttonAction = new ActionListener(){
            public void actionPerformed(ActionEvent e){
                if(e.getSource() == addVen){
                    f.dispose();
                    addRecord(uvms.loadRecord(fname));
                }
                else if(e.getSource() == exitPage){
                    //Terminates frame and returns to login page (main)
                    f.dispose();
                    uvms.staff_menu();
                }
            }
        };
        addVen.addActionListener(buttonAction);
        exitPage.addActionListener(buttonAction);

        table.getSelectionModel().addListSelectionListener(new
ListSelectionListener(){
            public void valueChanged(ListSelectionEvent e){
                if(e.getValueIsAdjusting()){
                    String id = table.getValueAt(table.getSelectedRow(),
0).toString();
                    f.dispose();
                    editRecord(uvms.loadRecord(fname), id);
                }
            }
        });

    }

    public void addRecord(final Vector<String> records){
        /*This function adds a new venue record in the system. */
        int cnt = uvms.countRecord("venue.txt");
        temp = new String[5];
        temp[0] = Integer.toString(cnt+1);

        /*Create JComponents for Columns*/
        JComponent[] ColumnData = new JComponent[4];
        //Right column - data
        final JTextField blockID = new JTextField();
        final JTextField venueName = new JTextField();
        final JComboBox cbox = new JComboBox(venueType);
        cbox.setFont(recFont);
        final JTextField descrp = new JTextField();

        //Create Jlabels for left column - fields
        for(int j=0; j<4; j++){
            ColumnData[j] = new JLabel(fieldOption[j], JLabel.RIGHT);
        }

        //Display New Frame
        final JFrame f = gui.createFrame("New Venue Record");

        //Add Component into Columns
        JPanel p = gui.createVertPanel(4,2,5,10);
        p.setPreferredSize(new Dimension(275,170));
        p.add(ColumnData[0]); p.add(blockID);
        p.add(ColumnData[1]); p.add(venueName);
        p.add(ColumnData[2]); p.add(cbox);
        p.add(ColumnData[3]); p.add(descrp);

        //Add buttons for users
        final JButton saveRec = new JButton("Add");
        final JButton cancelAdd = new JButton("Cancel");
        JPanel buttonPanel = gui.createHoriPanel();
        buttonPanel.add(saveRec);
        buttonPanel.add(cancelAdd);

        JPanel boxPanel = gui.createBoxPanel();
        boxPanel.add(p);

        JPanel formatSpaceVert = new JPanel();
        formatSpaceVert.setLayout(new BoxLayout(formatSpaceVert,
BoxLayout.X_AXIS));
        formatSpaceVert.add(Box.createVerticalStrut(50));

        JPanel formatSpaceHori = new JPanel();
        formatSpaceHori.setLayout(new BoxLayout(formatSpaceHori,
BoxLayout.X_AXIS));
        formatSpaceHori.add(Box.createHorizontalStrut(40));

        f.add(formatSpaceVert, BorderLayout.NORTH);
        f.add(formatSpaceHori, BorderLayout.EAST);
        f.add(boxPanel, BorderLayout.CENTER);
        f.add(buttonPanel, BorderLayout.SOUTH);
        f.setVisible(true);

        final int tempindex = cnt+1;
        ActionListener buttonAction = new ActionListener(){
            public void actionPerformed(ActionEvent e){
                if(e.getSource() == saveRec){
                    /*User wants to save a new record. Retrieve entered
strings in JTextfield and save it in temp[] */
                    temp[1] = blockID.getText();
                    //check if venue ID already exists
                    boolean checkExist = false;
                    for(int i=0; i<tempindex-1; i++){
                        String[] temp2 = new String[5];
                        temp2 = records.get(i).split("\t");
                        if(temp[1].equals(temp2[1])){
                            checkExist = true;
                            break;
                        }
                    }
                    if(!checkExist){ //if Venue ID does not exist, retrieve
other fields and save
                        temp[2] = venueName.getText();
                        //Venue Type is a dropdown box (JComboBox)
                        if(cbox.getSelectedIndex()==0)
                            temp[3] = "Tutorial Room";
                        else if(cbox.getSelectedIndex()==1)
                            temp[3] = "Lecture Room";
                        else if(cbox.getSelectedIndex()==2)
                            temp[3] = "Lecture Hall";
                        else
```

```
                             temp[3] = "Laboratory";
                         temp[4] = descrp.getText();
                         //Update file
                         recStatus = true;
                         updateRec(records, tempindex);
                         f.dispose();
                         venue_menu();
                     }else
                         JOptionPane.showMessageDialog(null, "ID already
exists!", "Warning", JOptionPane.WARNING_MESSAGE);
                 }else if(e.getSource() == cancelAdd){
                     //Add Process is cancelled
                     recStatus = false;
                     f.dispose();
                     venue_menu();
                 }
             }
         };
         saveRec.addActionListener(buttonAction);
         cancelAdd.addActionListener(buttonAction);

         allRecord = records;
         return;
     }

     public void editRecord(final Vector<String> records, final String id){
         /*This function allows admin to edit venue information.
          * Editable fields are venue name, venue type, description
          */
         int index = -1;

         //Check if record exists and saves the index of record within the
vector
         for(int i=0; i<records.size(); i++){
             temp = new String[0];
             temp = records.get(i).split("\t");
             if(temp[1].equals(id)){
                 index = i; //save index to replace the record (in vector)
after editing
                 break;
             }
         }

         JComponent[] ColumnData = new JComponent[4];

         //Create JComponents for right column (data)
         final JLabel blockID = new JLabel(temp[1], JLabel.CENTER);
         blockID.setFont(recFont);
         final JTextField venueName = new JTextField(temp[2]);
         final JComboBox cbox = new JComboBox(venueType);
         cbox.setFont(recFont);
         cbox.setSelectedItem(temp[3]);
         final JTextField descrp = new JTextField(temp[4]);
         //Create JComponents for left column (field)
         for(int j=0; j<4; j++){
             ColumnData[j] = new JLabel(fieldOption[j], JLabel.RIGHT);
         }
```

```
        //Display Edit Frame
        final JFrame editf = gui.createFrame("Edit Record");

        JPanel p = gui.createVertPanel(4,2,5,10);
        p.setPreferredSize(new Dimension(275,170));
        //Add Component into Columns
        p.add(ColumnData[0]);
        p.add(blockID);
        p.add(ColumnData[1]);
        p.add(venueName);
        p.add(ColumnData[2]);
        p.add(cbox);
        p.add(ColumnData[3]);
        p.add(descrp);

        //Create Buttons
        final JButton saveEdit = new JButton("Save");
        final JButton delVen = new JButton("Delete");
        final JButton cancelEdit = new JButton("Cancel");
        JPanel buttonPanel = gui.createHoriPanel();
        buttonPanel.add(saveEdit);
        buttonPanel.add(delVen);
        buttonPanel.add(cancelEdit);

        JPanel boxPanel = gui.createBoxPanel();
        boxPanel.add(p);

        JPanel formatSpaceVert = new JPanel();
        formatSpaceVert.setLayout(new BoxLayout(formatSpaceVert,
BoxLayout.X_AXIS));
        formatSpaceVert.add(Box.createVerticalStrut(50));

        JPanel formatSpaceHori = new JPanel();
        formatSpaceHori.setLayout(new BoxLayout(formatSpaceHori,
BoxLayout.X_AXIS));
        formatSpaceHori.add(Box.createHorizontalStrut(40));

        editf.add(formatSpaceVert, BorderLayout.NORTH);
        editf.add(formatSpaceHori, BorderLayout.EAST);
        editf.add(boxPanel, BorderLayout.CENTER);
        editf.add(buttonPanel, BorderLayout.SOUTH);
        editf.setVisible(true);

        final int tempindex = index;
        ActionListener buttonAction = new ActionListener(){
            public void actionPerformed(ActionEvent e){
                if(e.getSource() == saveEdit){
                    /*User wants to save edit. Retrieve strings in the
JTextfield and transfer into temp[] */
                    temp[2] = venueName.getText();
                    //Venue Type is a dropdown list (JComboBox)
                    if(cbox.getSelectedIndex()==0)
                        temp[3] = "Tutorial Room";
                    else if(cbox.getSelectedIndex()==1)
                        temp[3] = "Lecture Room";
                    else if(cbox.getSelectedIndex()==2)
```

```
                        temp[3] = "Lecture Hall";
                    else
                        temp[3] = "Laboratory";
                    temp[4] = descrp.getText();

                    //Update file
                    recStatus = true;
                    updateRec(records, tempindex);
                    editf.dispose();
                    venue_menu();
                }else if(e.getSource() == delVen){
                    deleteRecord(records, id);
                    editf.dispose();
                    venue_menu();
                }else if(e.getSource() == cancelEdit){
                    recStatus = false;
                    editf.dispose();
                    venue_menu();
                }
            }
        };
        saveEdit.addActionListener(buttonAction);
        delVen.addActionListener(buttonAction);
        cancelEdit.addActionListener(buttonAction);
    }

    public void updateRec(Vector<String> records, int index){
        /*This function saves all changes made to the records (new record
and edited record) */
        String str = new String();
        String msg;
        if(recStatus){
            for(int i=0; i<temp.length; i++)
                str += temp[i] + "\t";
            try{ //Update vector
                records.set(index, str);
                msg = new String("Record has been updated successfully!");
            }catch(ArrayIndexOutOfBoundsException arrE){
                //OutOfBounds: index is a new record
                records.addElement(str);
                msg = new String("Record has been added successfully!");
            }

            //Write vector into file
            try{
                writeFile("venue.txt", records);
                JOptionPane.showMessageDialog(null, msg, "Success",
JOptionPane.PLAIN_MESSAGE);
            }catch(IOException ioe){
                JOptionPane.showMessageDialog(null, "Operation failed!",
"Error", JOptionPane.ERROR_MESSAGE);
            }
            allRecord = records;
        }
    }

    public String[][] viewRecord(Vector<String> records){
        /*This function lets admin view all available venue information in
the system. */
        //Load each record to 1 row, each field to 1 column
        String data[][] = new String[records.size()][4];
        for(int i=0; i<records.size(); i++){
            temp = new String[0];
            temp = records.get(i).split("\t");
            for(int j=0; j<4; j++){
                data[i][j] = temp[j+1];
            }
        }
        return data;
    }

    public void deleteRecord(Vector<String> records, String id){
        /*This function completely removes a venue information from the
file. */
        int index = -1, ans;

        //Check if the entered venue ID exists
        for(int i=0; i<records.size(); i++){
            temp = new String[0];
            temp = records.get(i).split("\t");
            if(temp[1].equals(id)){//if the record exists, save the index
of the record within the vector
                index = i;
                break;
            }
        }

        ans = JOptionPane.showConfirmDialog(null, "Do you want to delete "
+ temp[1] +"?", "Confirmation", JOptionPane.YES_NO_OPTION,
JOptionPane.INFORMATION_MESSAGE);

        if(ans == JOptionPane.NO_OPTION || ans ==
JOptionPane.CLOSED_OPTION){
            //User does not want to remove the displayed venue
            return;
        }
        else{
            //Remove the venue record from the file
            //.remove() returns the value removed from vector
            records.remove(index);

            //Update file after removal
            try{
                writeFile("venue.txt", records);
                JOptionPane.showMessageDialog(null, "Record is deleted
successfully!", "Success", JOptionPane.PLAIN_MESSAGE);
            }catch(IOException ioe){
                JOptionPane.showMessageDialog(null, "Error updating file.",
"Error", JOptionPane.ERROR_MESSAGE);
            }
            allRecord = records;
        }
    }
    //searchRecord must search by file id
    public boolean searchRecord(int input, Vector<String> records){
        /*This function checks if the given input (file ID) exists */
        boolean found = false;
        for(int i=0; i<records.size(); i++){
            temp = new String[0];
            temp = records.get(i).split("\t");
            if(temp[0].equals(Integer.toString(input))){
                found = true;
                break;
            }
        }
        return found;
    }

    public void writeFile(String fname, Vector<String> allRecord) throws
IOException{
        /*This function updates the vector to the given file name
(venue.txt) */
        FileWriter writeFile = new FileWriter(fname, false);
        String[] temp = new String[0];
        for(int i=0; i<allRecord.size(); i++){
            temp = allRecord.get(i).split("\t");
            for(int j=0; j<temp.length; j++)
                writeFile.write(temp[j] + "\t");
            writeFile.write("\r\n");
            //write in new line, prepare for next record
        }
        writeFile.close();
    }
}
```

GUI.java

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

/*This class mainly provides formatting for JComponents
        to make sure the format throughout the program is consistent.*/
public class GUI{
    private Font heading = new Font("Calibri", Font.BOLD, 24);

    public JFrame createFrame(String pageName){
        JFrame f = new JFrame(pageName);
        f.setSize(500,375);
        f.setLayout(new BorderLayout());
        //Create JFrame at center of screen (Windows)
        f.setLocationRelativeTo(null);

        f.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
        //If user wants to close the frame
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                int ans = JOptionPane.showConfirmDialog(null, "Do you want
to exit the program?", "Confirmation", JOptionPane.YES_NO_OPTION,
JOptionPane.QUESTION_MESSAGE);
                if(ans == JOptionPane.YES_OPTION) System.exit(0);
            }
        });

        return f;
    }

    public JLabel BottomMessage(){
        JLabel l = new JLabel("Click on any record to perform action",
JLabel.CENTER);
        l.setForeground(Color.BLUE);
        l.setFont(new Font("Calibri", Font.BOLD, 18));
        return l;
    }

    public JPanel createHoriPanel(){
        JPanel p = new JPanel();
        p.removeAll();
        p.validate();
        p.setLayout(new FlowLayout());
        p.setMaximumSize(new Dimension(500,200));
        return p;
    }

    public JPanel createVertPanel(int row, int column, int hgap, int vgap){
        JPanel p = new JPanel();
        p.removeAll();
        p.validate();
        p.setLayout(new GridLayout(row, column, hgap, vgap));
        p.setMaximumSize(new Dimension(250,150));
        //p.setMaximumSize(new Dimension(420,230));
        return p;
    }

    public JPanel createBoxPanel(){
        JPanel p = new JPanel();
        p.removeAll();
        p.validate();
        p.setLayout(new FlowLayout());
        return p;
    }

    public JLabel createHeadingLabel(String text){
        JLabel l = new JLabel(text, JLabel.CENTER);
        l.setFont(heading);
        return l;
    }

    public JFrame createViewPage(String pageName){
        JFrame v = new JFrame(pageName);
        v.setSize(1000, 500);
        v.setLayout(new BorderLayout());
        v.setLocationRelativeTo(null);
        v.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
        v.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                int ans = JOptionPane.showConfirmDialog(null, "Do you want
to exit the program?", "Confirmation", JOptionPane.YES_NO_OPTION,
JOptionPane.QUESTION_MESSAGE);
                if(ans == JOptionPane.YES_OPTION) System.exit(0);
            }
        });

        return v;
    }
}
```