RoSys Program Manual

# Introduction

This manual is a guide that should help clarify some of the ideas we used throughout the program. It will detail how some parts of the program works.

An important thing to keep in mind about our program is that some parts, such as the user interface, were made purely to demonstrate the programs functionality. The main part of the program was the “Core” project, which is meant to serve as a backend server to any/multiple user exposed interfaces.

Our focus with the Core is the ability to integrate into other systems, so we made sure to use interfaces for some of the main resources with the program.

This focus on being able to integrate the program, as well as wishing to demonstrate the programs functions on its own, was a greater challenge than expected.



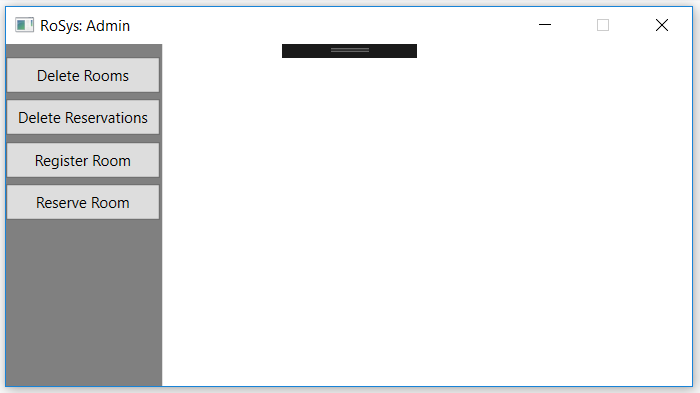
Picture

# The User Interface

Remember: The User interface was made purely to demonstrate features of our program

## Initial Run

When running the program, there are, initially, 3 options, as seen in Picture 1. This is essentially our “login”, where selection is made of which type of user will interact with the program.



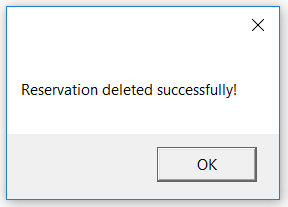
Picture

The different types of users have different functionality. Student and Teacher generally have the same features, but Teacher has access to rooms that Student does not.

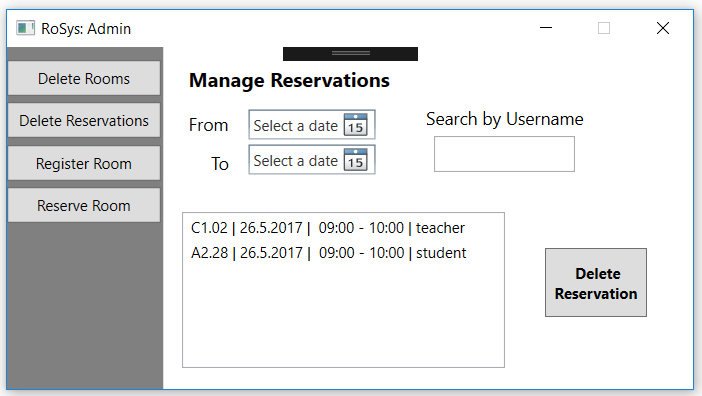
Admin has some management features, like deleting rooms and reservations, registering new rooms, selecting specific rooms for a reservation and access to a wider variety of rooms to reserve.

By pressing the “Delete Rooms” button, in Picture 2, a list of rooms appears, from which a selection can be made, and then can be deleted, Picture 3. This will delete Rooms from the repository and database. Also, it will delete all reservations for that room from the program and database.

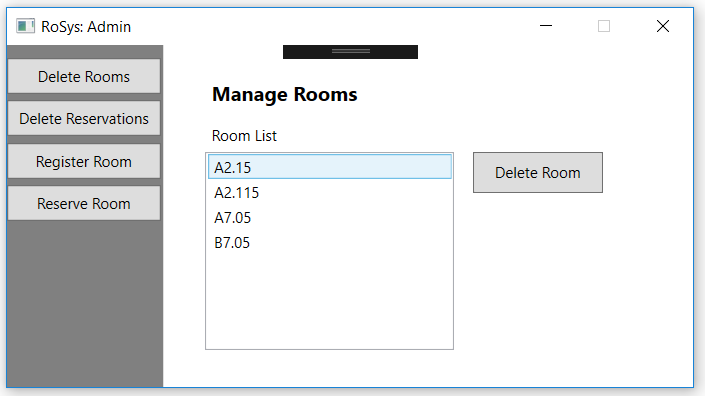
When the “Delete Reservations” button is pressed, a page, “Manage Reservations”, appears, Picture 4. If both select dates are empty as well as “Search by Username” the list shows all reservations. If only “Search by Username” is filled, then the list shows all reservations for that specific user. If only one “Select a date” field is filled with a specific date, then the list shows either all reservations from that date or until that date, depending on which date was filled in – from or to. If both “Select a date” fields are filled, then the list shows all reservations for that period. If only one “Select a date” field is filled with a specific date and “Search by Username” is filled, then the list shows either that users reservations from that date or until that date, depending on which date was filled in, the same as previously. And if all fields are filled in, then the list displays all reservations for that specific user in that specific period. If an admin selects a reservation and presses the “Delete Reservation” button that reservation will be removed from the repository and database and a message will appear, Picture 5. To close the message box, the admin needs to press the “OK” button.



Picture 5

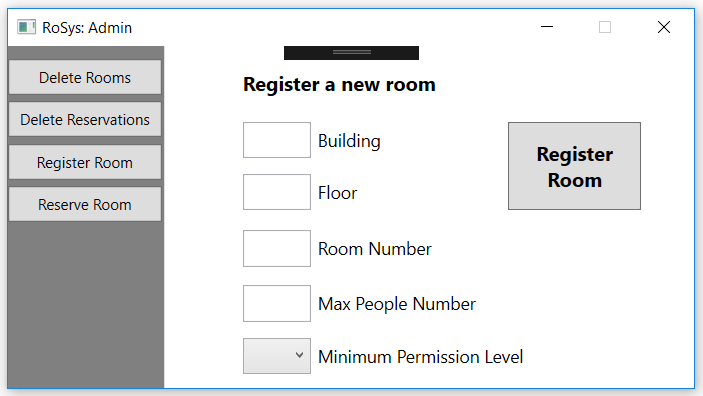


Picture 4



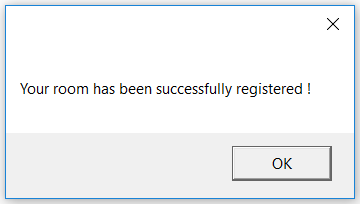
Picture 3

To register a new room in RoSys, an admin needs to press the “Register Room” button, Picture 6. Before pressing the “Register Room” button in the “Register a new room” page, all fields must be filled, otherwise the program will crash. In the “Building” field, only one letter can be inputted. If all fields are filled and the “Register Room” button is pressed, a message will popup, Picture 7. To close the message box, the admin needs to press the “OK” button.

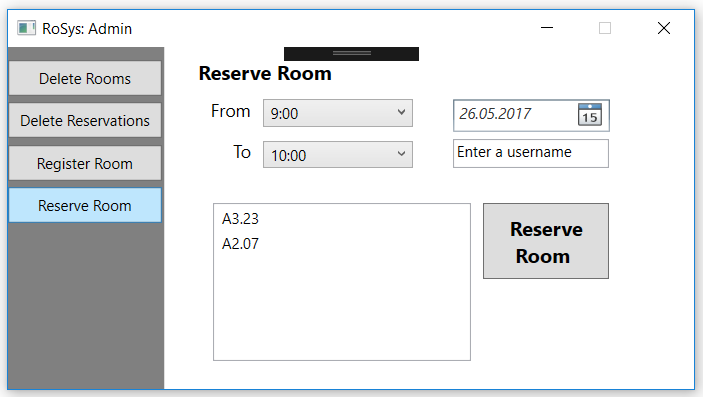


Picture 6

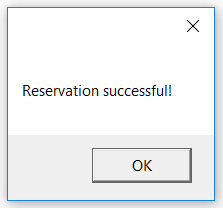
Admins can also reserve rooms. All fields must be filled. Date and times are automatically set to today’s date and 9:00-10:00, respectively. The user needs to exist in the database and repository and the text, “Enter a username”, must be cleared before the admin can input the username. The admin also needs to select which room he wants the reservation to be in. If everything is done right, then a message will appear, like in Picture 9.



Picture 7



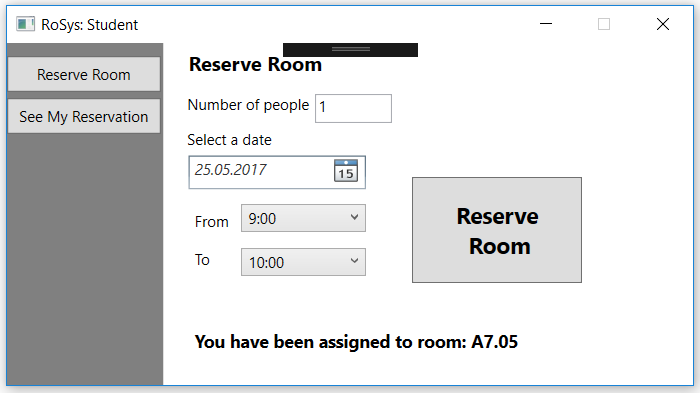
Picture 8



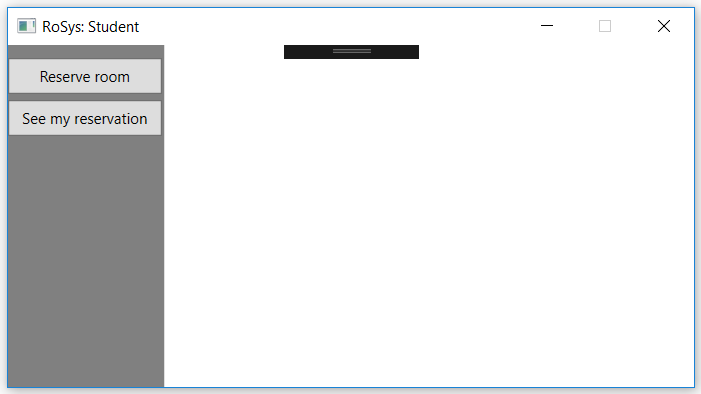
Picture 9

By pressing the “Student” button in the menu, Picture 1, the “RoSys: Student” window pops up, Picture 10.

The “Reserve Room” button opens a page with four parameters to reserve a room: “Number of people”, “Select a date”, “From” and “To” (Picture 11). The number of people that will be in the room needs to be typed in the textbox. The “Select a date” box is automatically set to today’s date if not changed manually by clicking on calendar icon and selecting a specific date. The “From” and “To” drop downs are made to select the time that the reservation will start and the time it will end. If any of the fields are still empty, then a message should be shown, but it has not been implemented yet. If a student user tries to reserve another room at the same time and date, the room will not be reserved and a message will be displayed, Picture 12. If there are available rooms with matching parameters as inputted, then a message saying that the user has been assigned to a room, as shown at the bottom of Picture 11, if there are no available rooms, then a message saying that there are no available rooms will appear, Picture 13.



Picture 11



Picture 10

To see the reservation the student user has made, he or she needs to press the button, “See My Reservation”, Picture 14. One list item shows one reservation. A reservation is shown as the room it is in, the date, the time and the user who made the reservation. If the user no longer needs the reservation, then he or she can select a reservation and press the button, “Delete Reservation”. This will delete the reservation from the repository and database.

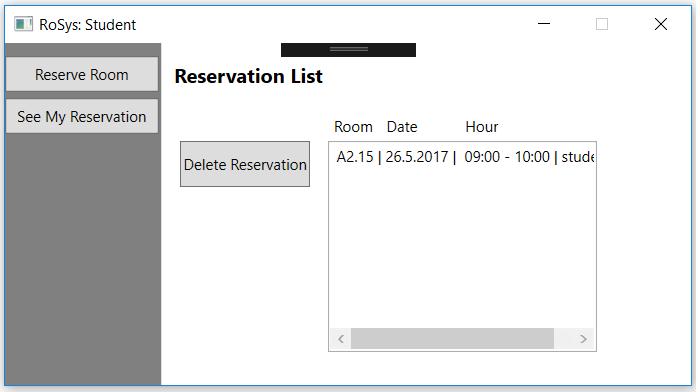


Picture 12



Picture 13

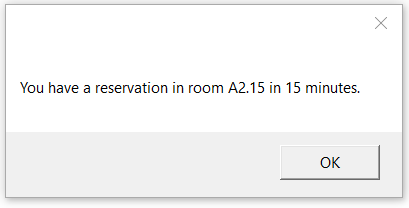
As we mentioned earlier, the teacher user has the same functions as the student user, but teachers can reserve rooms with teacher permission level, which student users are not able to reserve.



Picture 14

The system also notifies the user if they have a reservation in 15 minutes, Picture 15. By pressing button “OK” message box will be closed.

­­



Picture 15

# Code

This section will cover some of the features of the code.

## Core: DAL Facade

This class is like a bridge between our DAL and Core. It also converts things from database strings into objects and vice versa.

### DeleteAllUsers, DeleteAllRooms

Calls a method in DAL.Users or DAL.Rooms to delete all users or rooms from the database.

### InsertRoom, InsertUser, PassReservationToDAL

Calls a method in DAL.Rooms, DAL.Users or DAL.Reservations to insert a room, user or reservation to the database.

### GetUser, GetRoom, GetReservation

These three methods are the ones that collect data from the DAL and then return a User, Room or Reservation object to its caller.

### GetAllUsers, GetAllReservations, GetAllRooms

These three methods are like the previous ones, but they return lists of all User, Room or Reservation objects from the database.

### ConvertFromStringsToUserObjects

This method creates User objects from dictionaries of strings. Before creating the object, it converts the PermissionLevel string into an integer, so in Helper Functions this can be converted into permission level. Then it creates a new User and adds it to the list and returns the list.

### ConvertFromStringsToRoomObjects

This method creates Room objects from dictionaries of strings. Before creating the object, it converts the permission level string into an integer, so in Helper Functions this can be converted into permission level. It also converts floorNr, nr and peopleNr from string to integer and building from a string to char. Then it creates a new Room and adds it to the list and returns the list.

### ConvertFromStringsToReservationObjects

This method creates Reservation objects from dictionaries of strings. Before creating the object, it creates a new User using the username from the database and calls the Get(IUser) in the UserRepository to get the User from the repository. It also creates a new Room using the building, floorNr and nr from the database and calls the Get(IRoom) in the RoomRepository to get the Room from the repository. Then it creates a new Reservation and adds it to the list and returns the list.

### DeleteReservation, DeleteRoom, DeleteUser

Calls a method in DAL.Reservations, DAL.Rooms or DAL.Usersto delete a reservation, room or user from the database.

### ConvertFromReservationObjectToStrings

This method does the opposite of ConvertFromStringsToReservationObjects. It converts all the properties of the reservation to strings and adds them to a dictionary of strings.

## Core: HelperFunctions

Used to keep various generic functions that are used throughout the core project.

### ConvertIntToPermission

The method uses a switch to convert an int to a Permission type (0 = Student, 1 = Teacher, 2 = Admin)

### TimeCollides

Checks if a specific DateTime value lies between two other DateTime values.

## Core: Initialize

### StartUp

Initializes the program. All loading from the database is done here. Threads for notifications and checking the Change table start running here.

## Core: Reservation

### Equals

Reservations are considered equal if they both have the same User, From and To values. The comparison of those three are done using each ones respective Equals method.

### GetHashCode

The hash code of a reservation is the hash code of its User, From and To added together.

### To String

The ToString method returns the Room ID/In Queue (depending on if it is the queue or not) followed by the date of the reservation, the time of the reservation and finally, the username associated with the reservation.

## Core: ReservationRepository

The reservation repository is a singleton. It has list of reservations.

### RequestReservation

First, it checks if the user already has a reservation in the specified timeperiod. Then, it gets all rooms that have a permission level that is lower than or equal to the users permission level and have a higher peopleNr than the reservation specified. RemoveUnavailableReservations is called to return a list of rooms that do not have reservations in the specified timeperiod. If the list of available rooms is empty, then an exception is thrown and the reservation is added to the queue. If there are rooms in the list, then it creates a new Reservation, adds it to the repository and returns the room that the reservation will be in.

### GetAvailableRooms

Calls the GetPossible method in the RoomRepository to get the rooms the provided user has access to. RemoveUnavailableReservations is called to return a list of rooms that do not have reservations in the specified timeperiod. Then, the list is returned.

### RemoveUnavailableRooms

Receives a list of rooms and two DateTime values and calls the IsAvailable method in each room and sends the two DateTime values. If IsAvailable returns true the room is added to the list. Then, the list is returned.

### DeleteFromQueue

Removes the reservation from the queue.

### GetQueue

Returns the list of reservations in the queue.

### LoadFromDatabase

Adds the reservation to the repository from the database at startup. It also adds the reservation to the Room and User in the reservation

### DeleteFromRepository

Remove the reservation from the repository and not the database. It also removes the reservation from the Room and User in the reservation

### Clear

Removes all reservations from the repostitory and database.

### Add

Adds the reservation to the repository and database. It also adds the reservation to the Room and User in the reservation

### Delete

Removes the reservation from the repository and database. It also removes the reservation from the Room and User in the reservation. Then, it checks if there are any reservations in the queue that could replace the deleted reservation. Then a message alerts the user that they now have a reservation and the reservation is removed from the queue.

### Get

Returns the reservation(s) based on the parameters that were sent.

## Core: Reservations Observer

### Update

Sets the message property that is used for the notification.

## Core: Room

### IsAvailable

Checks if the room has a reservation between the DateTime value parameters, if not, true is returned, if it does, false is returned.

### Equals

Rooms are considered equal if they both have the same ID. The comparison of the IDs is done using the ID’s Equals method.

### GetHashCode

Returns the hash code of the room’s ID.

### ToString

Returns the room’s ID.

### CompareTo

Returns 1 if the other object’s MaxPeople is lower than this object’s MaxPeople, 0 if the MaxPeople are the same and -1 if the other object’s MaxPeople is greater than this object’s MaxPeople.

### AddReservation

Adds the reservation to the list of reservations in the Room object.

### GetReservations

Returns the list of reservations in the Room object.

### DeleteReservation

Removes a reservation from the list of reservations in the Room object.

## Core: RoomRepository

### Clear

Removes all rooms from the repository and database. It also calls the Clear method in the ReservationRepository.

### Add

Adds the room to the repository and the database.

### LoadFromDatabase

Adds the room to the repository from the database at startup.

### DeleteFromRepository

Removes the room from the repository.

### Get

Returns the room(s) based on the parameters that were sent.

### GetPossible

Returns a list of rooms based on the parameters sent.

### Delete

Removes the room from the repository and database. It also removes any reservation inside that room.

## Core: RosysThreads

Implements the IObservable interface.

### NotificationThread

Notifies the observer that there is a reservation in 15 minutes. It checks the reservations every minute.

### MaintenanceThread

Removes reservations that have passed from the repository and the queue.

### CheckChangeTable

Gets information from the Change table and determines which command was used in the database and calls the respective method.

### InsertInformation

Determines which table the insert command was used on and calls the respective method.

### AddUserToRepository

Calls the DALFacade to get the user that was added from the database and add it to the repository.

### AddRoomToRepository

Calls the DALFacade to get the room that was added from the database and add it to the repository.

### AddReservationToRepository

Calls the DALFacade to get the reservation that was added from the database and add it to the repository.

### UpdateInformation

Not Implemeneted.

### DeleteInformation

Determines which table the delete command was used on and calls the respective method.

### DeleteUserFromRepository

First, it creates a new user using the primary key value in the change table. Then, it calls the Get(IUser) method and deletes the returned user from the repository.

### DeleteRoomFromRepository

First, it creates a new room using the primary key value in the change table. Then, it calls the Get(IRoom) method and deletes the returned room from the repository.

### DeleteReservationFromRepository

First, it creates a new room and user using the primary key value in the change table. Then, it creates a new reservation using the username, dateTo and dateFrom. Then, it calls the Get(Reservation) method and deletes the returned reservation from the repository.

### Subscribe

Adds the observer to the list of observers.

### Unsubscribe

Removes the observer from the list of observers.

### Notify

Calls the Update method in each Observer and sends the message to be outputted.

## Core: SystemSettings

The System Settings class is currently used to define the “environment” the program runs in.

We set up 3 levels of environment, but we only use 2: Development and Production currently have no differences, but Test affects which database we connect to.

### Update System Environment

This method updates the Environment variable.

## Core: User

### Equals

Users are considered equal if they both have the same ID. The comparison of the Usernames is done using the Username’s Equals method.

### GetHashCode

Returns the hash code for the user’s Username.

### GetReservation

Returns the list of reservations in the User object.

### AddReservation

Adds the reservation to the list of reservations in the User object.

### DeleteReservation

Removes a reservation from the list of reservations in the User object.

### HasReservation

Checks if the User has a reservation in the specified timeperiod.

## Core: UserRepository

### Clear

Removes all users from the repository and database. It also calls the Clear method in the ReservationRepository.

### Add

Adds the user to the repository and the database.

### LoadFromDatabase

Adds the user to the repository from the database at startup.

### DeleteFromRepository

Removes the user from the repository.

### Get

Returns the user(s) based on the parameters that were sent.

### Delete

Removes the user from the repository and database. It also removes any reservation inside that user.

## UI.GUI: LoggedIn

Is the class that stores which user is currently logged in. It is used by the GUI to keep track of which user was initially selected.

## DAL: Change

### GetAllChangesFromDatabase

Runs the stored procedure, SP\_GetAllChanges, in the database. Then, it adds all the values of one row to a dictionary with the key being the column name. Next, the dictionary is added to a list. Finally, the list is returned.

### DeleteChangeFromDatabase

Runs the stored procedure, SP\_DeleteChange, in the database using the id parameter.

### DeleteAllChangesFromDatabase

Runs the stored procedure, SP\_DeleteAllChanges, in the database.

## DAL: Database

### OpenConnection

Creates an SQLConnection and calls the Open method.

### CloseConnection

Calls the Close method for the SQLConnection.

## DAL: DatabaseConn

A factory class to switch which connection string is used.

## DAL: ConnProd

Makes the connection string be to the production database.

## DAL: ConnTest

Makes the connection string be to the production database.

## DAL: Reservations

### GetAllReservationsFromDatabase

Runs the stored procedure, SP\_GetAllReservations, in the database. Then, it adds all the values of one row to a dictionary with the key being the column name. Next, the dictionary is added to a list. Finally, the list is returned.

### GetReservationFromDatabase

Runs the stored procedure, SP\_GetReservation, in the database using the id as the parameter. Then, it adds all the values of one row to a dictionary with the key being the column name. Next, the dictionary is added to a list. Finally, the list is returned.

### DeleteReservationFromDatabase

Runs the stored procedure, SP\_DeleteReservation, in the database using the username, from and to parameters.

### StoreReservationIntoDatabase

Runs the stored procedure, SP\_StoreReservationIntoDatabase, in the database using the properties of a reservation as parameters.

## DAL: Rooms

### GetAllRoomsFromDatabase

Runs the stored procedure, SP\_GetAllRooms, in the database. Then, it adds all the values of one row to a dictionary with the key being the column name. Next, the dictionary is added to a list. Finally, the list is returned.

### GetRoomFromDatabase

Runs the stored procedure, SP\_GetRoom, in the database using the building, floorNr and nr as parameters. Then, it adds all the values of one row to a dictionary with the key being the column name. Next, the dictionary is added to a list. Finally, the list is returned.

### DeleteRoomFromDatabase

Runs the stored procedure, SP\_DeleteRoom, in the database using the building, floorNr and nr as parameters.

### DeleteAllRoomsFromDatabase

Runs the stored procedure, SP\_DeleteAllRooms, in the database.

### InsertRoomToDatabase

Runs the stored procedure, SP\_InsertRoom, in the database using the properties of a room as parameters.

## DAL: Users

### GetAllUsersFromDatabase

Runs the stored procedure, SP\_GetAllUsers, in the database. Then, it adds all the values of one row to a dictionary with the key being the column name. Next, the dictionary is added to a list. Finally, the list is returned.

### GetUserFromDatabase

Runs the stored procedure, SP\_GetUser, in the database using the username as the parameter. Then, it adds all the values of one row to a dictionary with the key being the column name. Next, the dictionary is added to a list. Finally, the list is returned.

### InsertUserToDatabase

Runs the stored procedure, SP\_InsertUser, in the database using the properties of a user as parameters.

### DeleteUserFromDatabase

Runs the stored procedure, SP\_DeleteUser, in the database using the username as the parameter.

### DeleteAllUserFromDatabase

Runs the stored procedure, SP\_DeleteAllUsers, in the database.