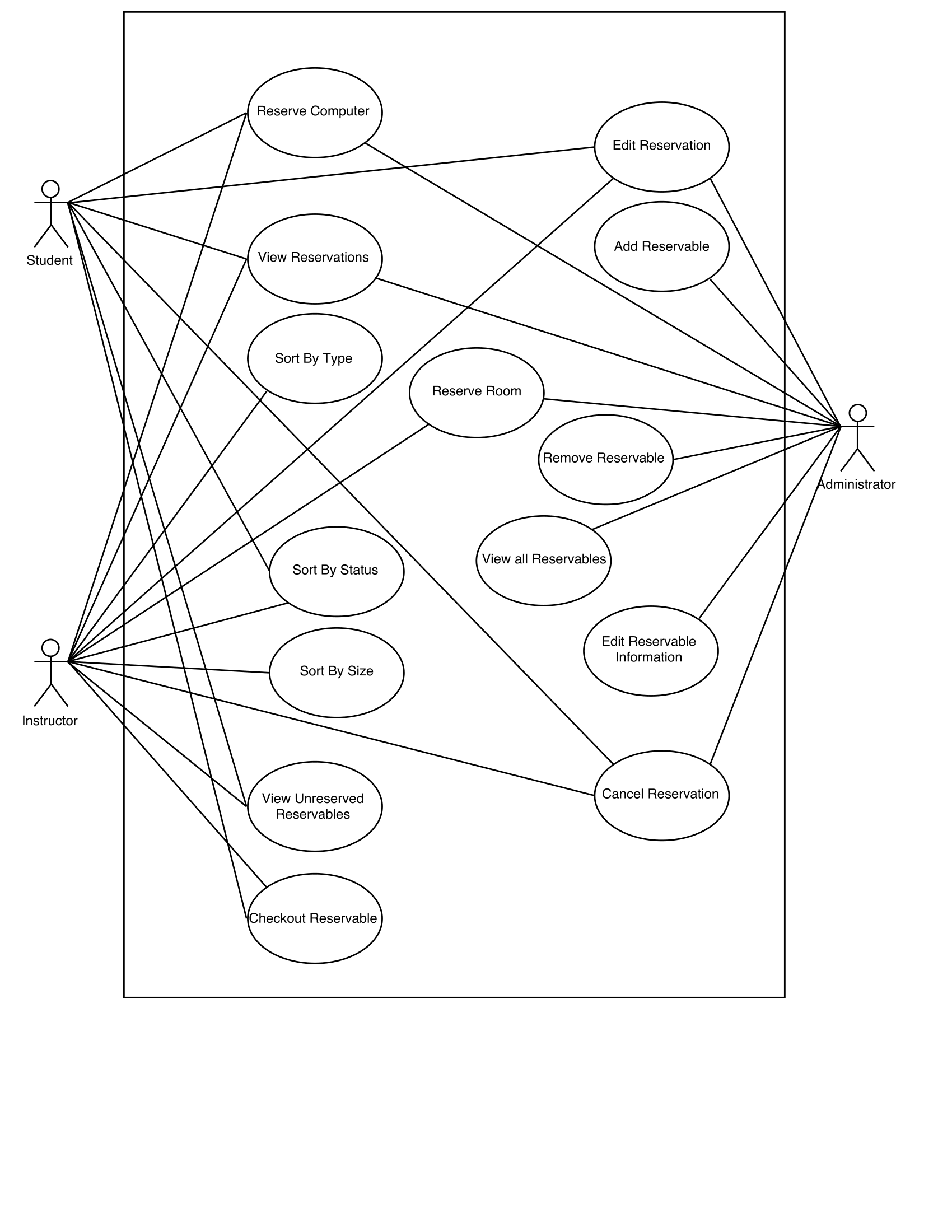
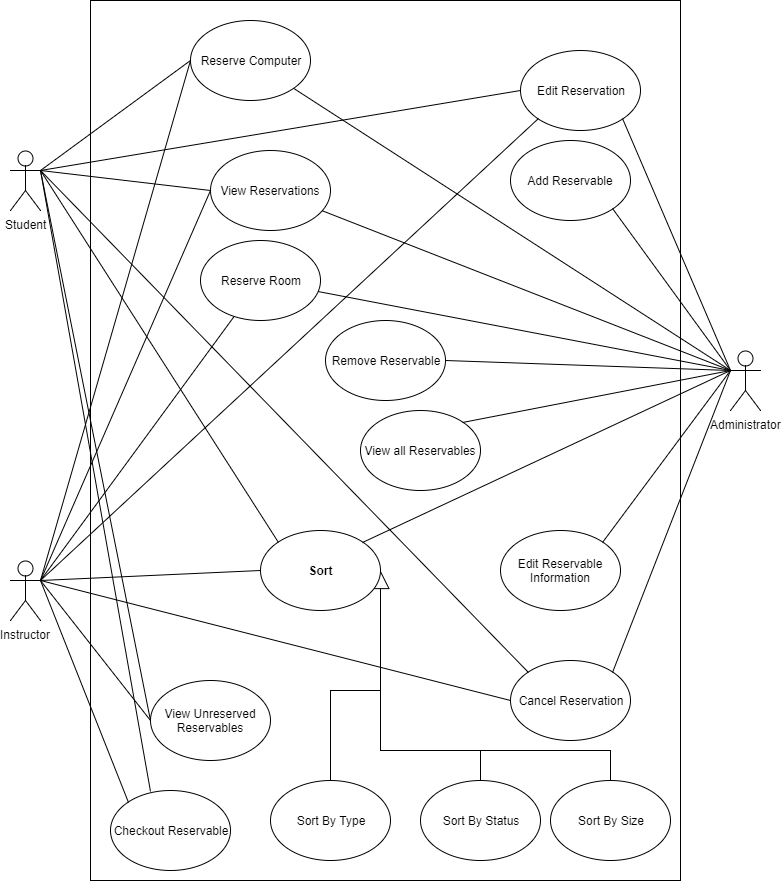
1. Deliverables

Initial use case diagram

Revised use case diagram



Use case specification cards:

|  |
| --- |
| Name: Sort by status |
| Actors: Student, Instructor, administrator |
| Trigger: User wants to sort the reservations by status |
| Preconditions: None |
| Post conditions: list is sorted by status |
| Success Scenario:   1. Users requests the reservation information to be sorted by status 2. System sorts the list based on the reserve status of the rooms 3. System displays the list to user |
| Alternatives flows: None |

|  |
| --- |
| Name: Sort by Size |
| Actors: Student, Instructor, administrator |
| Trigger: User wants to sort the reservations by Size |
| Preconditions: None |
| Post conditions: list is sorted by Size |
| Success Scenario:   1. Users requests the reservation information to be sorted by Size 2. System sorts the list based on the reserve status of the rooms 3. System displays the list to user |
| Alternatives flows: None |

|  |
| --- |
| Name: Sort by Type |
| Actors: Student, Instructor, administrator |
| Trigger: User wants to sort the reservations by Type |
| Preconditions: None |
| Post conditions: list is sorted by Type |
| Success Scenario:   1. Users requests the reservation information to be sorted by Type 2. System sorts the list based on the reserve status of the rooms 3. System displays the list to user |
| Alternatives flows: None |

|  |
| --- |
| Name: Cancel reservations |
| Actors: Student |
| Trigger: Student cancels a reservation |
| Preconditions: Student has a reservation |
| Post conditions: Computer’s Status is updated to reservable |
| Success Scenario:   1. Students selects a reservation they made and cancels the reservation, 2. Changing the status to reservable during the student’s original reservation. |
| Alternative Flows: Student Cannot cancel a reservation because they don’t have a reservation |
| Name: View reservations |
| Actors: Student |
| Trigger: Student selects Checkout resealable |
| Preconditions: Student selected Checkout resealable |
| Post conditions: Reservation list is displayed for Student |
| Success Scenario:   1. Reservation list is displayed to user |
| Alternative Flows: System has no rooms to view |

|  |
| --- |
| Reserve Room |
| Instructor |
| Instructor willing to use classrooms |
| Instructor requesting to reserve classrooms |
| System updates the status of reservation |
| 1. Instructor choose to request reserve for classrooms 2. System checks if there are available classrooms the selected timeslot 3. System shows the list of available classrooms with number of seats 4. Instructor choose the building and room that are available 5. System adds and update the reservation to the system |
| 2.1. There are available classrooms on selected timeslot  2.2. There are no available classrooms on selected timeslot |

|  |
| --- |
| Reserve Computer |
| Student, Instructor |
| Students or instructor willing to use the lab computers |
| Student or Instructor requesting to reserve the lab computers |
| System updates the status of reservation |
| 1. Instructors or students choose to request reserve for computer 2. System checks if there are available lab computers for specific timeslot 3. System shows the list of available lab computers 4. Instructors or students choose the lab and computer they want to reserve 5. System adds and update the reservation to the system |
| 2.1. There are available classrooms on selected timeslot  2.2. There are no available lab computers on selected timeslot |

|  |
| --- |
| Name: Edit Reservable Information |
| Actors: Administrator |
| Trigger: Administrator selects a reservable and chooses to edit it |
| Preconditions: There is a reservable selected that is preexisting in the system |
| Postconditions: The attributes and information of the reservable are changed |
| Success Scenario:   1. Administrator selects edit reservable 2. System uses <<include: View all Reservables>> 3. Administrator chooses reservable to edit 4. System “unlocks” attributes of that reservable 5. Administrator changes attributes and selects to save changes 6. System changes and saves attributes of the selected reservable 7. System displays list of reservables |
| Alternative Flows:  There are no reservables in the system  Administrator decides not to save changes and cancels edit |

|  |
| --- |
| Name: Remove Reservable |
| Actors: Administrator |
| Trigger: Administrator selects a reservable and chooses to remove it |
| Preconditions: There is a reservable selected that is preexisting in the system |
| Postconditions: The selected reservable is no longer in the system as a reservable |
| Success Scenario:   1. Administrator selects remove reservable 2. System uses <<include: View all Reservables>> 3. Administrator chooses reservable to remove 4. System removes reservable from list of reservables 5. System displays list of reservables with the deleted reservable no longer available |
| Alternative Flows: There are no reservables in the system |

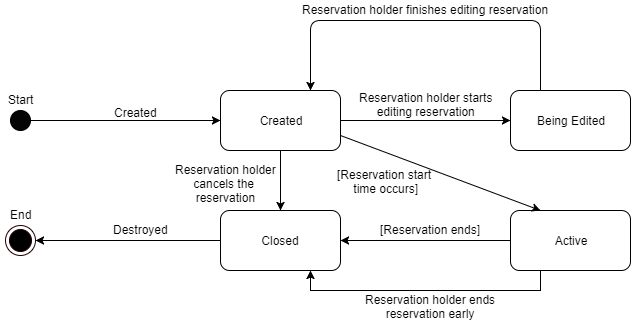
|  |
| --- |
| Name: Add Reservable |
| Actors: Administrator |
| Trigger: Administrator chooses to add a reservable |
| Preconditions: Reservables screen is active |
| Postconditions: A new reservable is added to the list of reservables |
| Success Scenario:   1. Administrator chooses to add a reservable 2. System creates a new reservable 3. Administrator fills in attributes of the reservable and selects save 4. System saves reservable attributes and adds reservable to list of reservables 5. System displays list of reservables |
| Alternative Flows: Administrator decides not to add reservable to system and cancels add |

|  |
| --- |
| Name: View Reservables |
| Actors: System |
| Trigger: Remove or Edit reservable is selected |
| Preconditions: System is running |
| Postconditions: Reservables list is displayed |
| Success Scenario:   1. System calls view reservables 2. System retrieves list of reservables 3. System Displays name of reservables in a list to user |
| Alternative Flows: None |

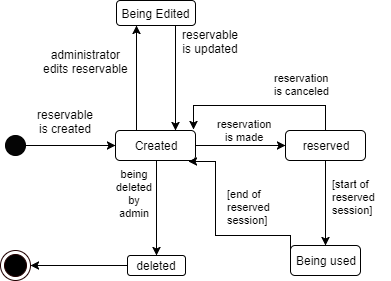
|  |
| --- |
| Name: Checkout Reservable |
| Actors: Student, Instructor |
| Trigger: User logs in to check out the reservable |
| Preconditions: The user has an account |
| Post conditions: Reservation has been removed from the user’s list, if it was there |
| Success Scenario:  1. User selects reservable to check out  2. User checks out reservable  3. System removes reservable from the user’s list |
| Alternative Flows:  1.1 If there are no reservables in the user’s list, then the system returns to the previous state |

|  |
| --- |
| Name: Edit Reservation |
| Actors: Student, Instructor, Administrator |
| Trigger: User selects to edit a reservation |
| Preconditions: The user has an account |
| Post conditions: The user has the same number of reservations as before the use case was performed |
| Success Scenario:  1. The system performs <include: View Reservations>  2. The user selects a reservation to edit  3. The system displays the reservation  4. The user selects to change the reservation room  5. The system preforms <include: View Unreserved Reservables>  6. The user selects the new reservation  7. The system adds the old reservation back to the list, and removes the newly selected one |
| Alternative Flows:  4.1 The user only changes the time, in which case the reservation time is modified, and no more steps are performed.  6.1 The user cancels the edit, in which case the system returns to its previous state |

Reservation state chart:



Reservable state chart:



CRC cards:

|  |  |
| --- | --- |
| Student | |
| Manages his or her computer reservation.  View reservation status of computers and classrooms.  Create reservation for computer.  Cancel his or her reservation.  View list of reservables | Reservation  Computer |
| Instructor | |
| Manages his or her reservation of computer or classroom.  View reservation status of computers and classrooms.  Create reservation for computer or classroom.  Cancel his or her reservation.  View a list of available reservables. | Reservation  Reservable |

|  |  |
| --- | --- |
| Administrator | |
| Manage all the reservations.  View all the reservations.  Add rooms and computers to the list.  Remove rooms and computer from the list.  Manage the list of students and Instructors.  Manage and view the reservable list | Reservation  Reservable |

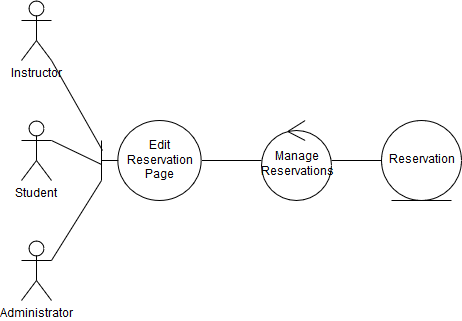
|  |  |
| --- | --- |
| Reservation | |
| Maintain reserved computer and classroom data.  Maintain reservation data  Maintain the available computer and classroom list by building and date/time.  Maintain the status of computer and classroom based on reservation time and duration. | Person  Reservation |

|  |  |
| --- | --- |
| Room | |
| Maintain the available times.  Maintain room data.  Check reservation availability.  Check reservation privileges.  Maintain a list of the reservations on the reservable  Display availability to users | Reservation  Instructor  Administrator |

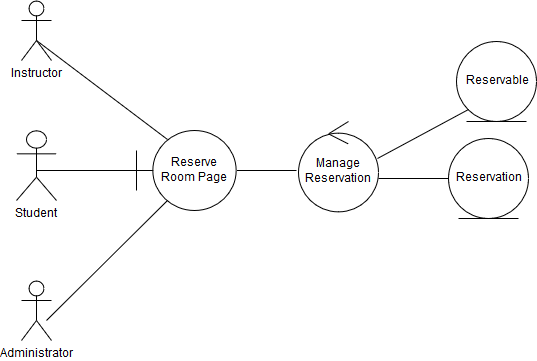
|  |  |
| --- | --- |
| Computer | |
| Maintain the available times.  Maintain computer data.  Check reservation availability.  Maintain a list of the reservations on the reservable  Display availability to users | Reservation  Student  Instructor  Administrator |

Robustness Diagrams:

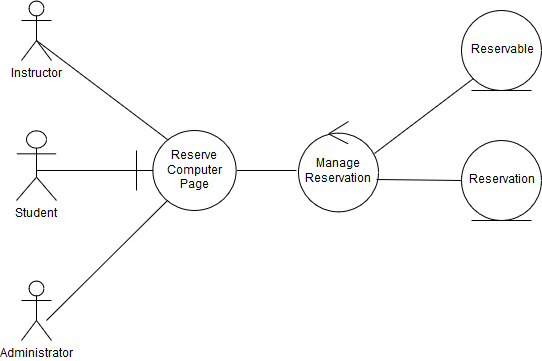
View Reservations:



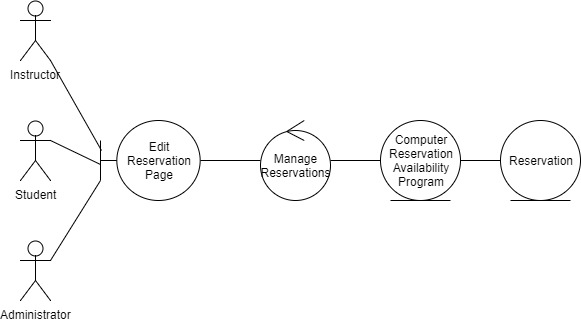
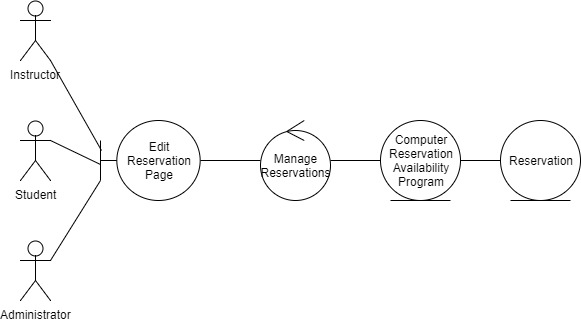
Reserve Room:



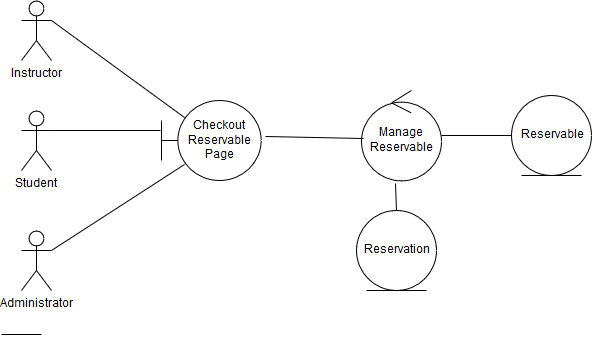
Reserve Computer:



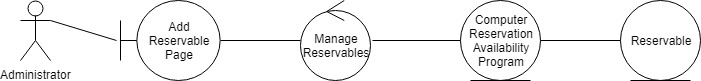
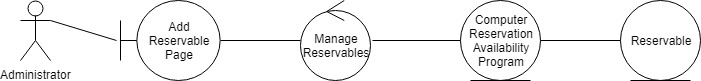
Edit Reservation:



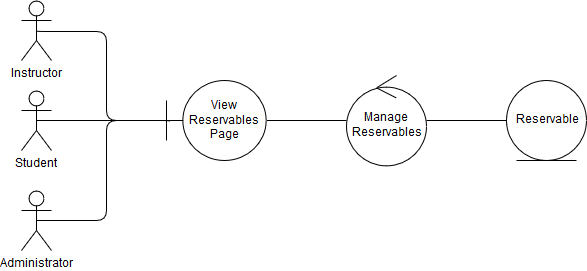
Checkout Reservable:



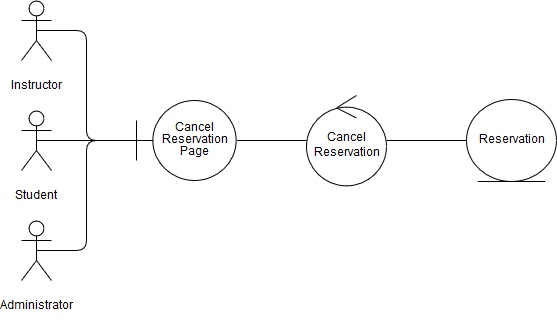
Add Reservable:



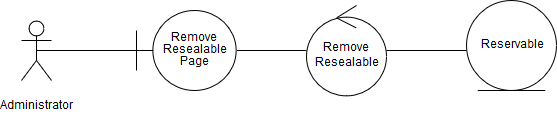
View Reservables:



Cancel Reservation:



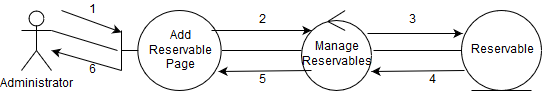
Remove Reservable:

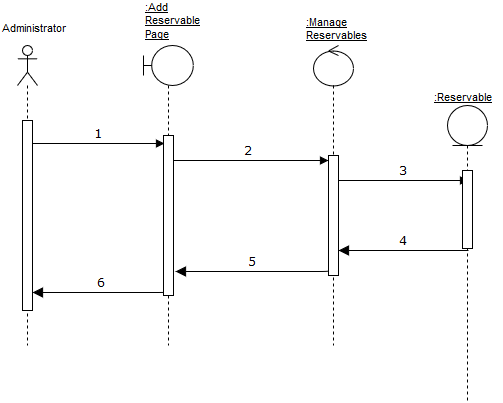


Collaboration/Sequence Diagrams:

Add Reservable:

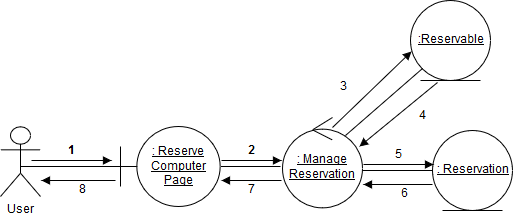
1. Provide reservable information
2. Pass reservable information
3. Create reservable
4. Return reservable
5. Reservable added
6. Reservable added

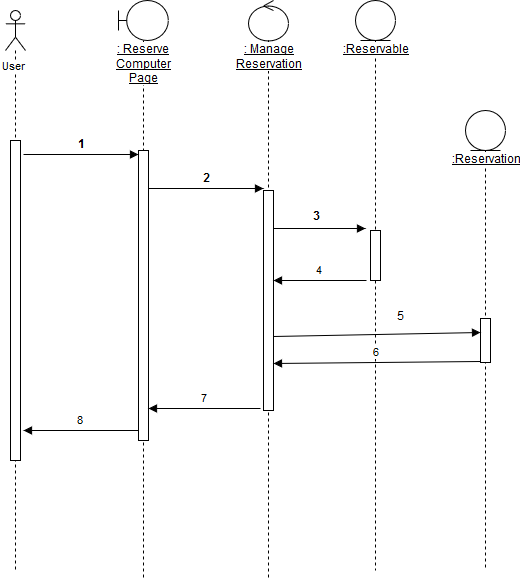




Reserve Computer:

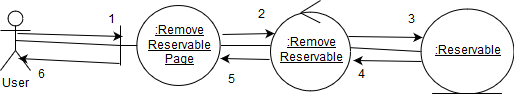
1. Request to reserve a computer
2. Pass reservation info
3. Is Reservable Available
4. Reservable Is Available
5. Create Reservation
6. Return Reservation
7. Reservation Added
8. Reservation Added

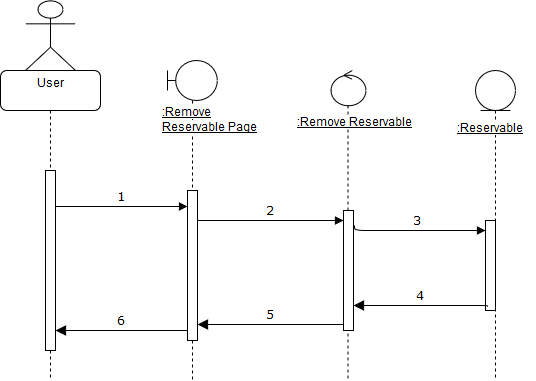




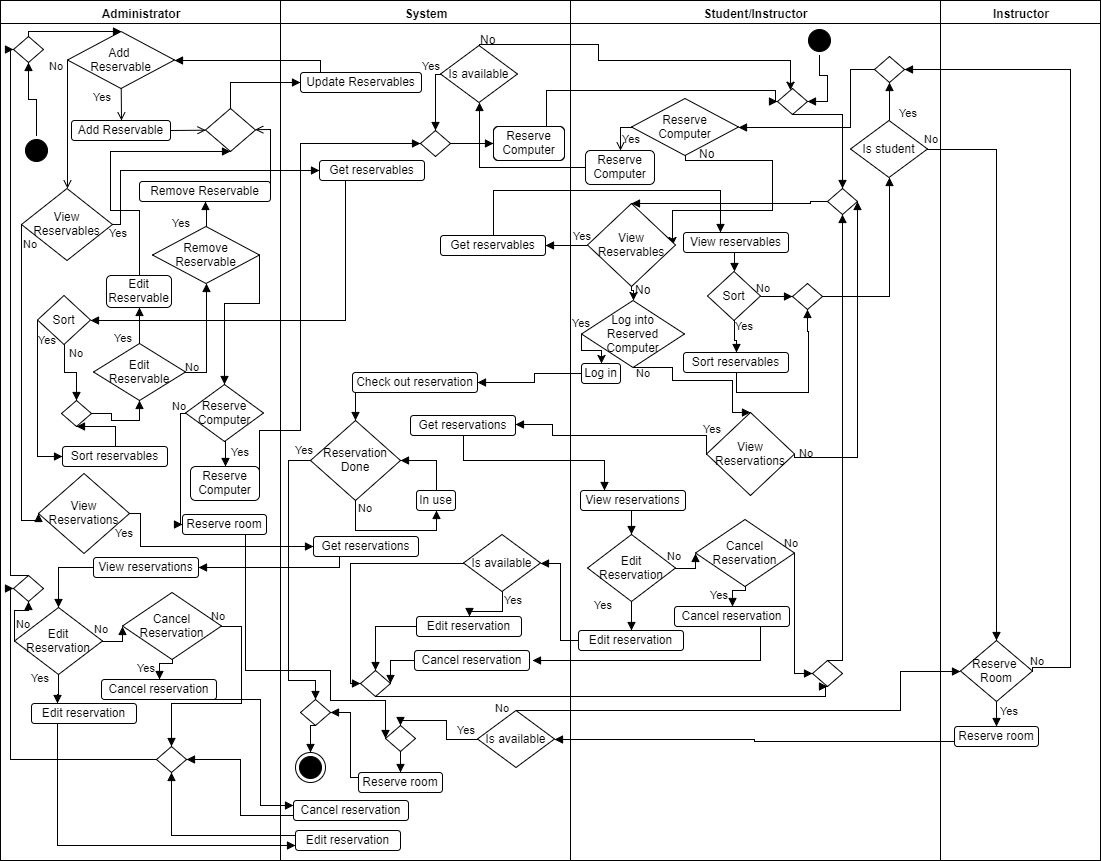
Remove Reservable:

1. Select the reservable to remove
2. Pass reservable information
3. Delete reservable
4. Reservable deleted
5. Successfully removed
6. Successfully removed

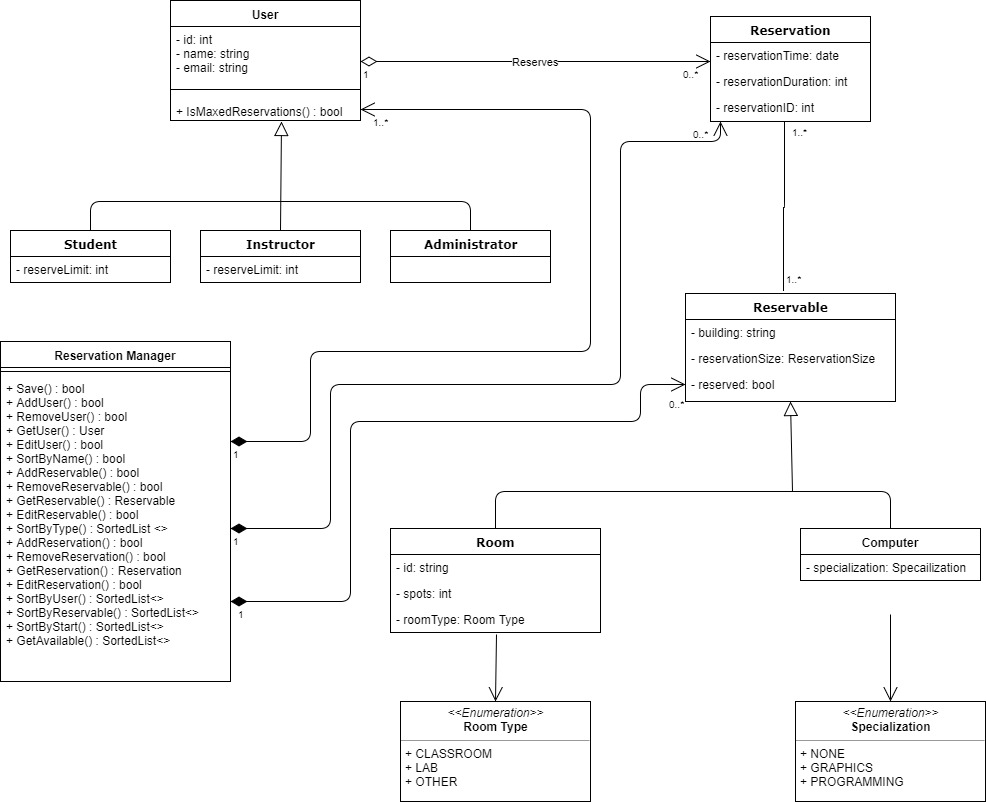




Activity Diagram:



Class Diagram:



B.

Our system was programed in C# Using the windows forms in visual studio. The system currently only runs on windows desktop computers. The information is currently stored in three text documents: reservables.txt, reservations.txt, and users.txt. The system currently only supports the main functionality and has a rudimentary GUI.

The functionality that the system currently supports includes administrators being able to manage reservables and users. It allows all users to check if a reservable is available for a specific time and duration, view and delete their current reservations. When managing reservables the administrator can create, delete, or edit reservables. When creating a computer reservable the admin can select what room to put it in. the admin can also move computers from one room to another. When managing users, the admin can create, edit, and delete users. The admin must supply the user type, name and email. The admin can also edit any of these fields for any user. When searching for reservations, only the reservables for that specific time appear (no browsing reservations).

The GUI currently is very basic. The reservables and users are only displayed by there ID number with no other information to signify what the object is. Reservations are displayed with basic information but not formatted well for user reference. The GUI is not resized or customized for different screens resolutions or window sizes.

Future improvement ideas or features that are yet to be implemented include a check-in/check-out of reservables so currently checked-out reservables can be viewed and if a reservation is ended early, the reservable can be checked out by someone else earlier. This would also include “walk-in” reservations for quick access. Another not yet implemented feature that was planned was displaying the type of reservable and location and information when selecting a reservable to reserve. This would also extend to users when the administrator is editing users in the database. Speaking of databases, the entire system would use databases to store the users, reservables, and reservations instead of using text files and reading from and writing to them on startup and shutdown respectively. A third feature would have included removing all reservations from the database when the reservable they were made for had been deleted. This would also send a message to the holder of any reservation of that reservable to notify them that the reservation that had been cancelled so they could reserve a different room or computer. A potential improvement on our current design could include adding laptops to the reservable items on a daily or an hourly rental schedule.

C.

|  |  |  |
| --- | --- | --- |
| Team Member | Total Time | Activities |
| Alex Bisbach | 16 hours | Activity Diagram, various other diagrams which were divided up throughout the semester, assisting in system design, requirements elicitation, and writing file IO code, and part of the presentation, and all the deliverables for this document |
| Ben Mehn | 12 hours | Specification cards, state charts, crc cards, use case diagram, and various other diagrams which were divided up throughout the semester, and the presentation |
| Evan Gjerde | 13 hours | Work on initial backend code, various diagrams, much of this document, system design, requirements |
| Jun Jeong | 12 hours | Work on various documents, system design, requirements, designing some of the GUI menus |
| Levon Swenson | 20 hours | Work on GUI menus, linking front and backends, and remaking much of the backend code, various diagrams, system design, requirements, and much of this document |

D.

One major difficulty was learning how to use C#. Some of us had never used C# before, so learning the differences between C# and C++ was sometimes a challenge, especially when it came to public and private constructors giving us problems with not being able to create things and us not knowing why for a bit.

Another difficulty was the inevitable finding time to meet. Matching up our schedules was difficult at times due to other class projects, work, and other obligations. This was solved by meeting to get started and distribute work to do when each of us had a chance. This allowed us to communicate the ideas and things we wanted to implement while still being able to work on the program when each of us did not feel rushed or like we should be working on something that was more urgent at that time.

This also allowed us to distribute the workload more evenly. This was a problem at the beginning of the project and sometimes lead to one person feeling overwhelmed or too busy, but the problem was more or less solved by the time the final 2 milestones were worked on and turned in.