Room Management System

**Requirements Analysis Document**

Version 4

CSCI 4711 Software Engineering

Spring 2023

Augusta University

Team Members

James Wilt

Charles Smith

Ian Roberson

**Table of Contents**

[1 INTRODUCTION 3](#_Toc132278520)

[1.1 SCOPE OF SYSTEM 3](#_Toc132278521)

[2 REQUIREMENTS OF SYSTEM 4](#_Toc132278522)

[2.1 FUNCTIONAL REQUIREMENTS 4](#_Toc132278523)

[2.2 NON-FUNCTIONAL REQUIREMENTS 5](#_Toc132278524)

[2.3 USE CASES 6](#_Toc132278525)

[2.4 USE CASE DESCRIPTIONS 7](#_Toc132278526)

[2.5 REQUIREMENTS ANALYSIS 13](#_Toc132278527)

[3 USER INTERFACE MOCKUPS 20](#_Toc132278528)

[3.1 STARTUP/LOGIN 20](#_Toc132278529)

[3.2 EMPLOYEE DASHBOARD 21](#_Toc132278530)

[3.3 ADMIN DASHBOARD 21](#_Toc132278531)

[3.4 RESERVE ROOM 22](#_Toc132278532)

[3.5 CANCEL RESERVATION 22](#_Toc132278533)

[3.6 LOGOUT 23](#_Toc132278534)

[4 OBJECT DESIGN 24](#_Toc132278535)

[4.1 OBJECT RELATIONSHIP 24](#_Toc132278536)

[4.2 DETAILED CLASS DESIGN 26](#_Toc132278537)

[4.3 DATABASE DESIGN 27](#_Toc132278538)

[5 SYSTEM DESIGN 28](#_Toc132278539)

[5.1 SUBSYSTEM DECOMPOSITION 28](#_Toc132278540)

[6 APPENDIX 29](#_Toc132278541)

[6.1 APPENDIX A-SOURCE CODE 29](#_Toc132278542)

# INTRODUCTION

## SCOPE OF SYSTEM

The Room Management System (RMS) is a software tool for the reservation of rooms. It includes actors such as *Employee* who represent the users who reserve rooms, and *Admin,* the administrators who can executively cancel existing reservations.

The system supports both actors by logging and storing all the information pertaining to the reservations that are made via the system. This is stored via an internal database. The customers are able to interact with the system via the RMS Graphical User Interface.

The system includes functionality for verification and authorization of users, making room reservations, and canceling reservations. All changes to room statuses are logged in the internal database to ensure the most accurate status of the rooms are given to the actors utilizing the interface.

# REQUIREMENTS OF SYSTEM

## FUNCTIONAL REQUIREMENTS

* Startup: This functionality performs the initial configuration of the system including its database and user interface.
* Login – All users of the system must be verified as either an employee or an admin before they can use the Room Management System. This function verifies the user, grants them access to other functionality, and saves each successful long to the database.
* ReserveRoom – Employees can choose what room to reserve via this function. The employee will be able to choose which building, room, date, and time to reserve.
* CancelReservation – This function lets admins cancel reservations made by employees. Admins will be able to see the building, room, date, and time along with which employee reserved them.
* Logout – This function allows users of the system to logout of their accounts when they have finished using the RMS. Following this function will be the login screen so different users can begin using the RMS while saving each logout to the database.

## NON-FUNCTIONAL REQUIREMENTS

* + - * Platform:
        + Target operating system is Microsoft Windows.
        + System is not web-based and should not utilize a web browser.
        + System should be implemented using the Java or C# programming language.
      * Security
        + All user input is validated for appropriateness and prevention of SQL injection attacks.
        + User passwords should not be stored as plaintext in the database.
      * Usability
        + System should implement a Graphical User Interface.

## USE CASES

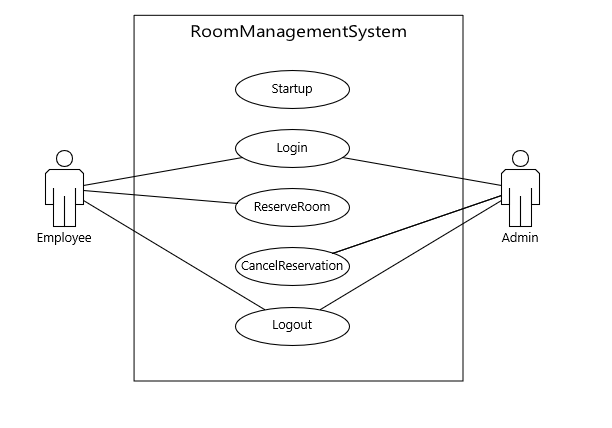


Figure 2.1: Use Case Diagarm

## USE CASE DESCRIPTIONS

|  |  |
| --- | --- |
| *Use case name* | Startup |
| *Participating*  *actors* |  |
| *Precondition* |  |
| *Flow of events* | 1. This use case is initiated when the application is launched. 2. **The RMS runs through its startup protocol, initializes the database and displays the GUI’s LoginMenu.** |
| *Post condition* | The LoginMenu is displayed. |
| *Security*  *requirements* | The Users Password is not stored as plaintext in the database. |

Figure 2.2: Startup

|  |  |
| --- | --- |
| *Use case name* | Login |
| *Participating*  *actors* | Initiated by the Employee and Admin |
| *Precondition* |  |
| *Flow of events* | 1. The user enters their username and password to login to the RMS. 2. **RMS receives the input and verifies the username and password are valid and authenticates the user against the database. Username and password are valid, and RMS displays the appropriate menu for the user. Login is saved to database.** |
| *Post condition* | The appropriate menu is displayed for the user |
| *Security*  *requirements* | Password should be obfuscated for the user.  User inputs should be verified against their specifications.  User inputs are validated to prevent SQL injection |

Figure 2.3: Login: Authorized User

|  |  |
| --- | --- |
| *Use case name* | Login |
| *Participating*  *actors* | Initiated by the Employee and Admin |
| *Precondition* |  |
| *Flow of events* | 1. The user enters their username and password to login to the RMS. 2. **RMS receives the input and verifies the username and password are valid and authenticates the user against the database. The username and password are invalid, and the GUI displays an error message.** |
| *Post condition* | The LoginMenu is displayed for the user |
| *Security*  *requirements* | Password should be obfuscated for the user.  User inputs should be verified against their specifications.  User inputs are validated to prevent SQL injection |

Figure 2.4: Login: Unauthorized User

|  |  |
| --- | --- |
| *Use case name* | ReserveRoom |
| *Participating*  *actors* | Initiated by Employee |
| *Precondition* | Employee is logged into the system. |
| *Flow of events* | 1. The Employee selects a building and room to reserve by pressing the “Reserve” button on the EmployeeDashboard. 2. **RMS presents the ReserveRoom window for the employee to choose the date and time for the reservation.** 3. The Employee selects the desired date and time to reserve, and then presses the “Make Reservation” button. 4. **RMS updates the database according to which building, room, date and time selected and presents the EmployeeDashboard.** |
| *Post condition* | The database is updated with room reservation information.  The EmployeeDashboard is displayed |
| *Security*  *requirements* |  |

Figure 2.5: ReserveRoom: Successful Reservation

|  |  |
| --- | --- |
| *Use case name* | CancelReservation |
| *Participating*  *actors* | Initiated by Admin |
| *Precondition* | Admin is logged into the system. |
| *Flow of events* | 1. Admin selects reservation to cancel using the “Cancel” button on the AdminDashboard 2. **RMS presents the CancelReservation menu with the details of what building, room, date and time were reserved by which employee.** 3. Admin confirms the room details and selects the “Cancel Reservation” button. 4. **RMS updates the database and displays the AdminDashboard**. |
| *Post condition* | The database is updated.  The AdminDashboard is displayed. |
| *Security*  *requirements* |  |

Figure 2.6: CancelReservation

|  |  |
| --- | --- |
| *Use case name* | Logout |
| *Participating*  *actors* | Initiated by Employee or Admin |
| *Precondition* | User is logged into the system. |
| *Flow of events* | 1. This function is activated when a user selects the “logout” button from the appropriate menu. 2. **RMS presents the LogoutMenu with a message and a “Close” button**. 3. The user selects the Close button. 4. **RMS returns the LoginMenu for another user to login. Logout is saved to the database.** |
| *Post condition* | The LoginMenu is displayed. |
| *Security*  *requirements* |  |

Figure 2.7: Logout

## REQUIREMENTS ANALYSIS

This page intentionally left blank

Diagram

Description automatically generated

Figure 2.7: Startup

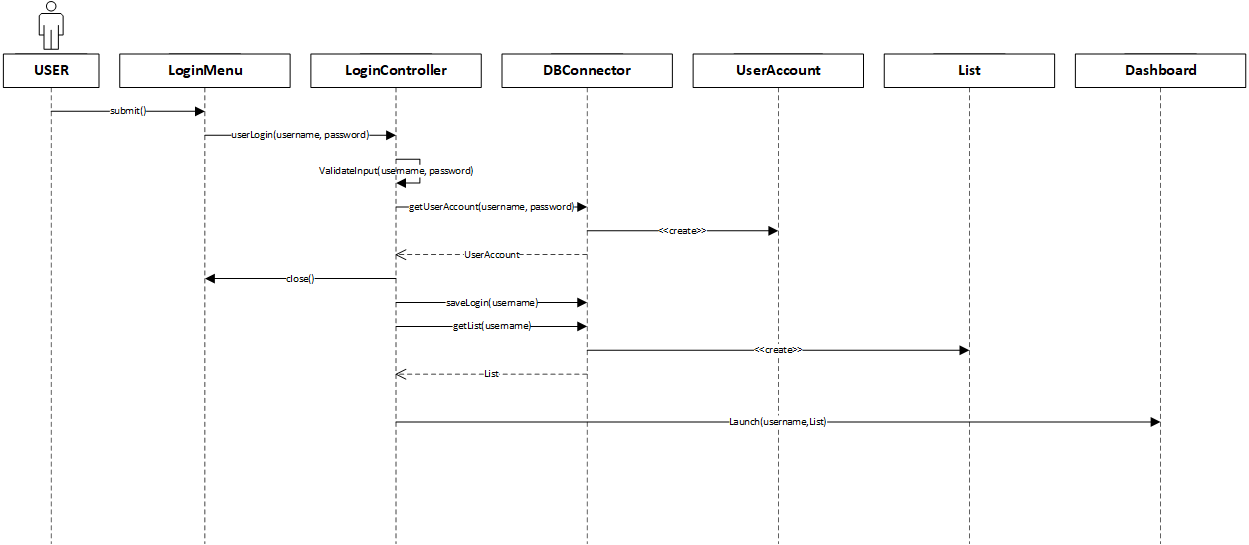


Figure 2.8: Login: Authorized User

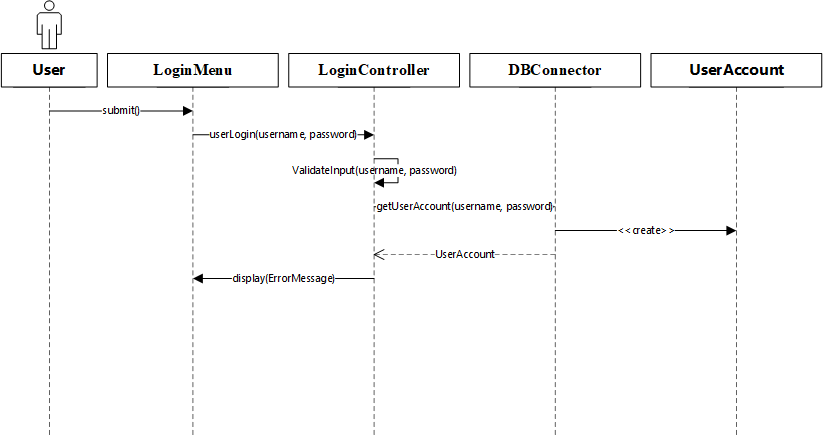


Figure 2.9: Login: Unauthorized User

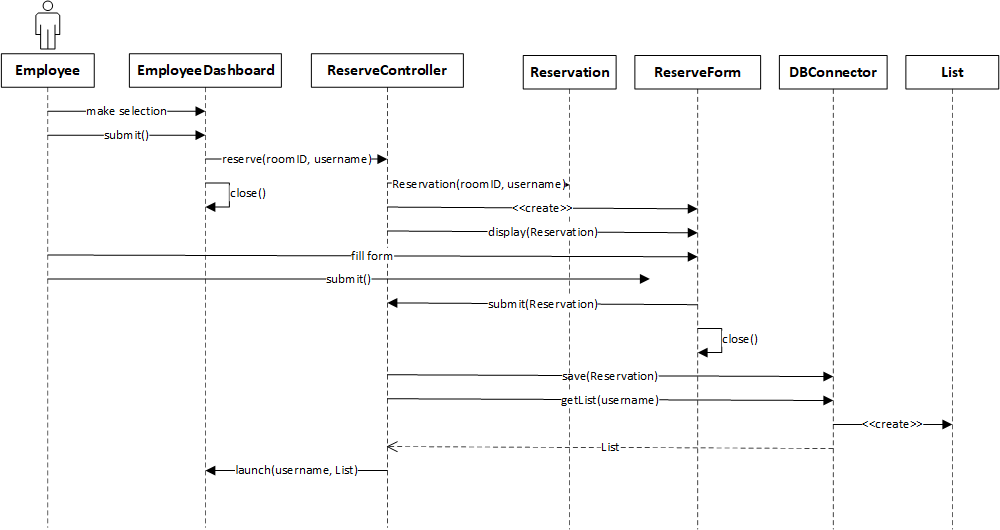


Figure 2.10 ReserveRoom

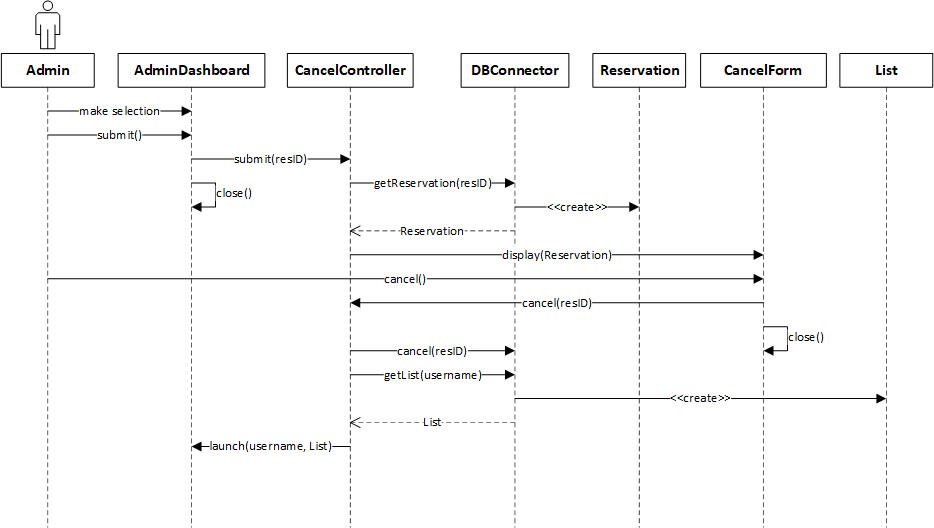


Figure 2.11 CancelReservation

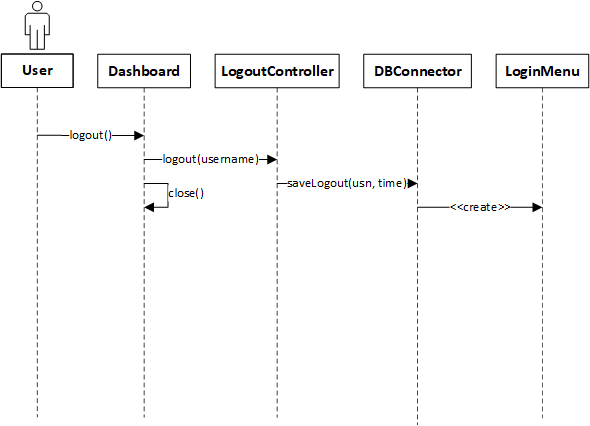


Figure 2.12 Logout

# USER INTERFACE MOCKUPS

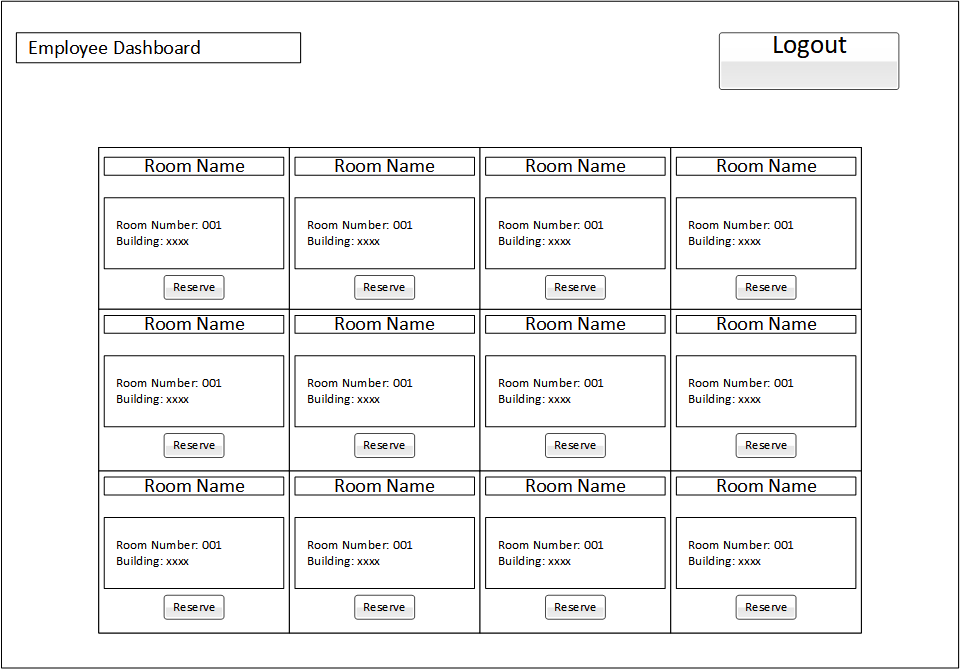
## STARTUP/LOGIN

Graphical user interface, text, application

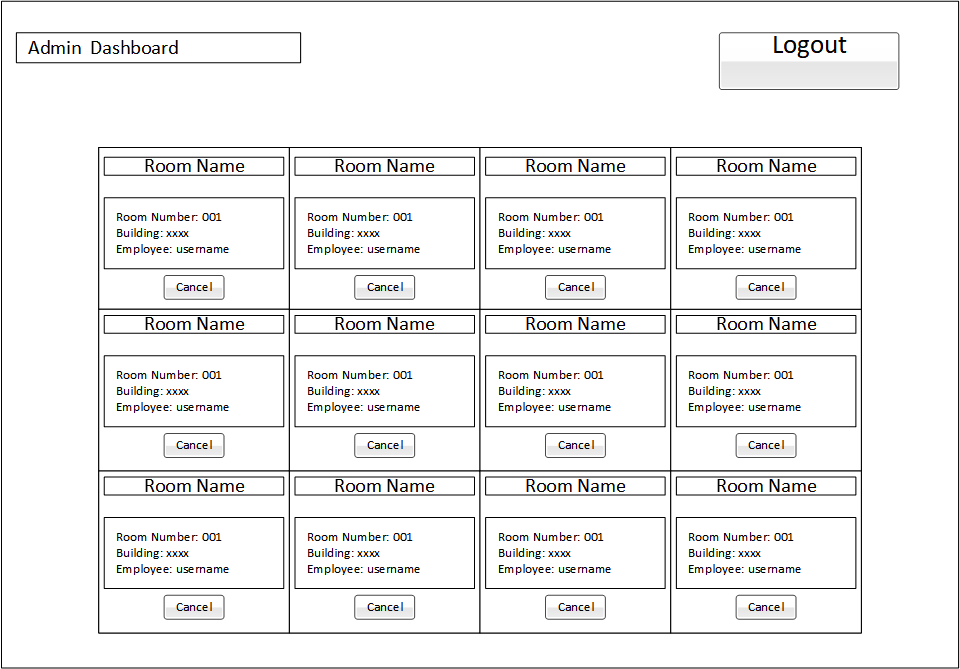
Description automatically generated

Figure 3.1: LoginMenu

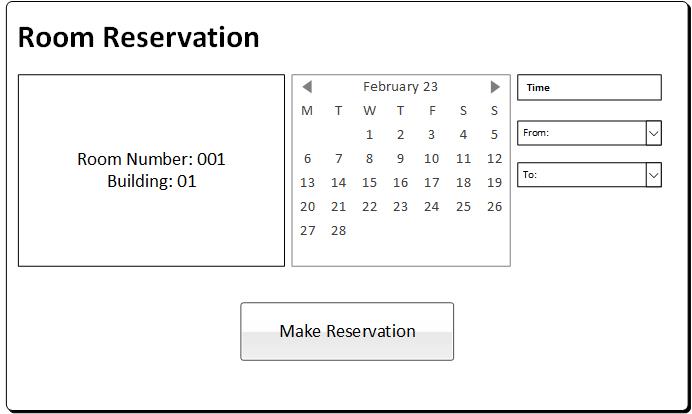
## EMPLOYEE DASHBOARD

  
Figure 3.2: EmployeeDashboard

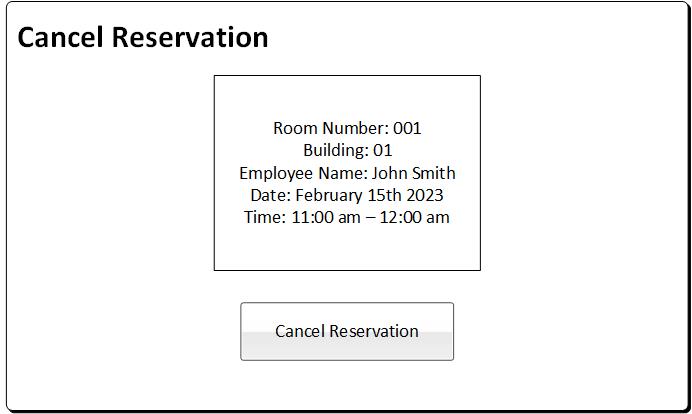
## ADMIN DASHBOARD

  
Figure 3.3: AdminDashboard

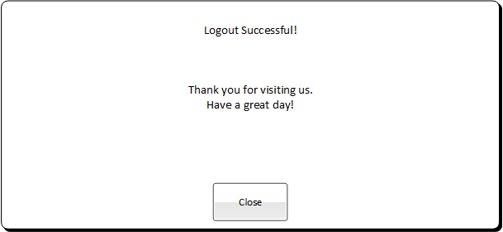
## RESERVE ROOM

  
Figure 3.4: ReserveRoom Menu

## CANCEL RESERVATION

  
Figure 3.5: CancelReservation Menu

## LOGOUT

  
Figure 3.6: Logout Menu

# OBJECT DESIGN

## OBJECT RELATIONSHIP

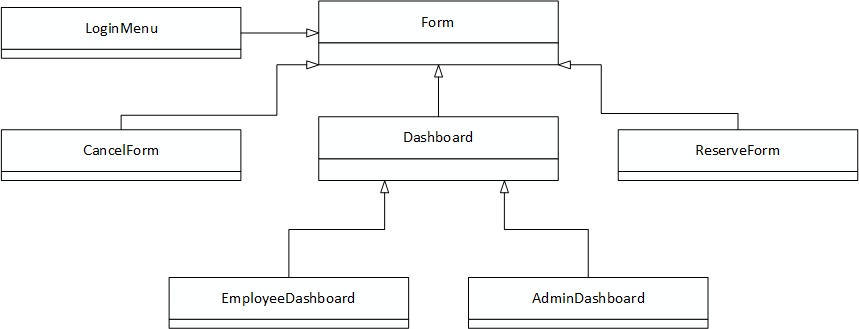
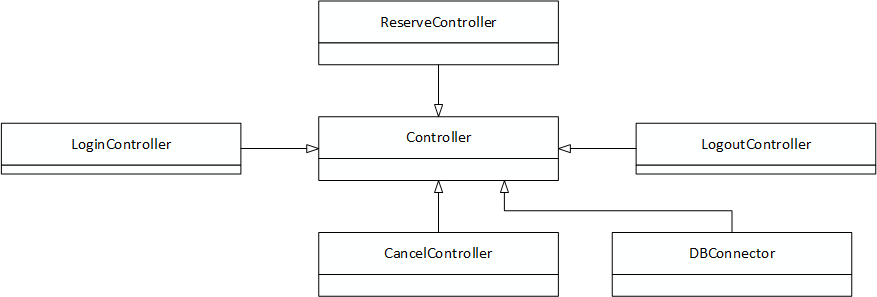


Figure 4.1: Class Diagram: Boundary

Figure 4.2: Class Diagram: Control

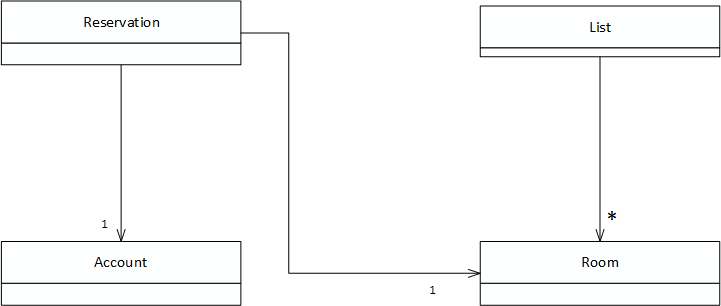


Figure 4.3: Class Diagram: Entity

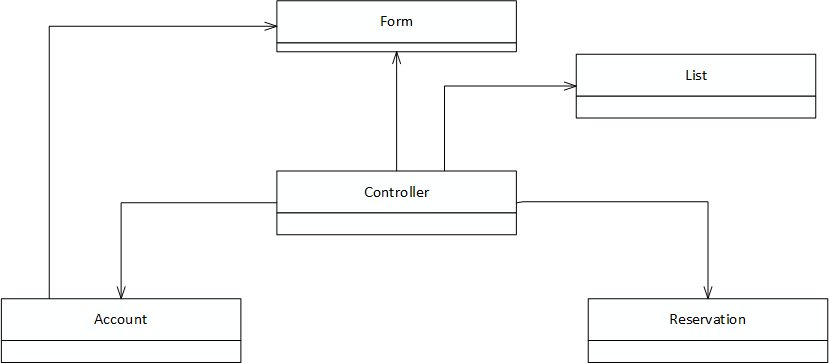


Figure 4.4: Class Diagram

## DETAILED CLASS DESIGN

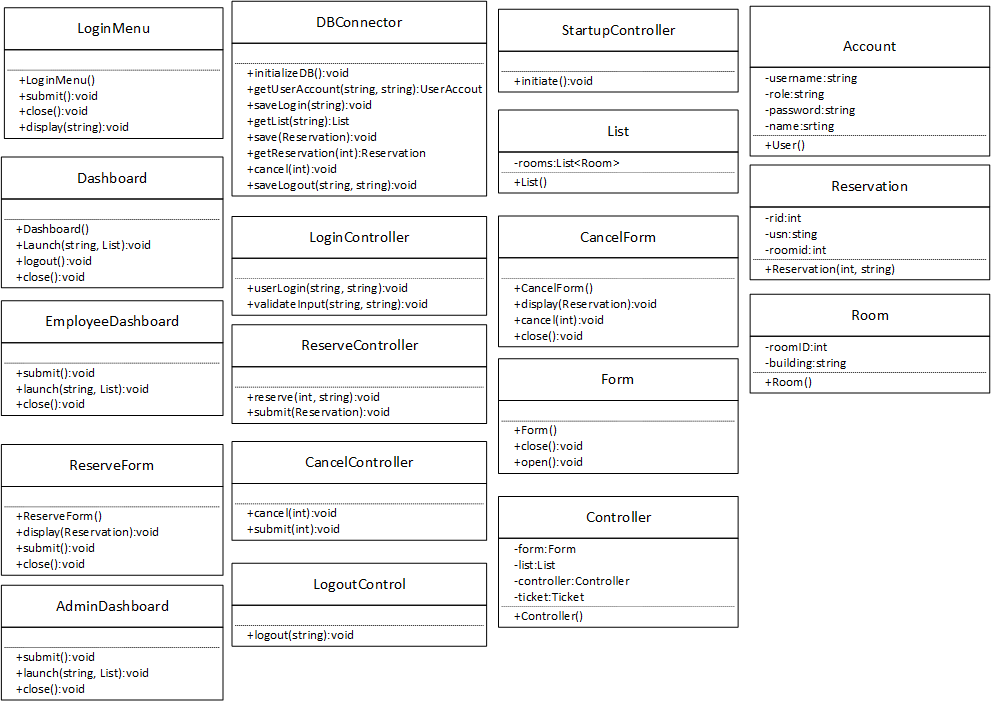


Figure 4.5: Detailed Diagram

## DATABASE DESIGN

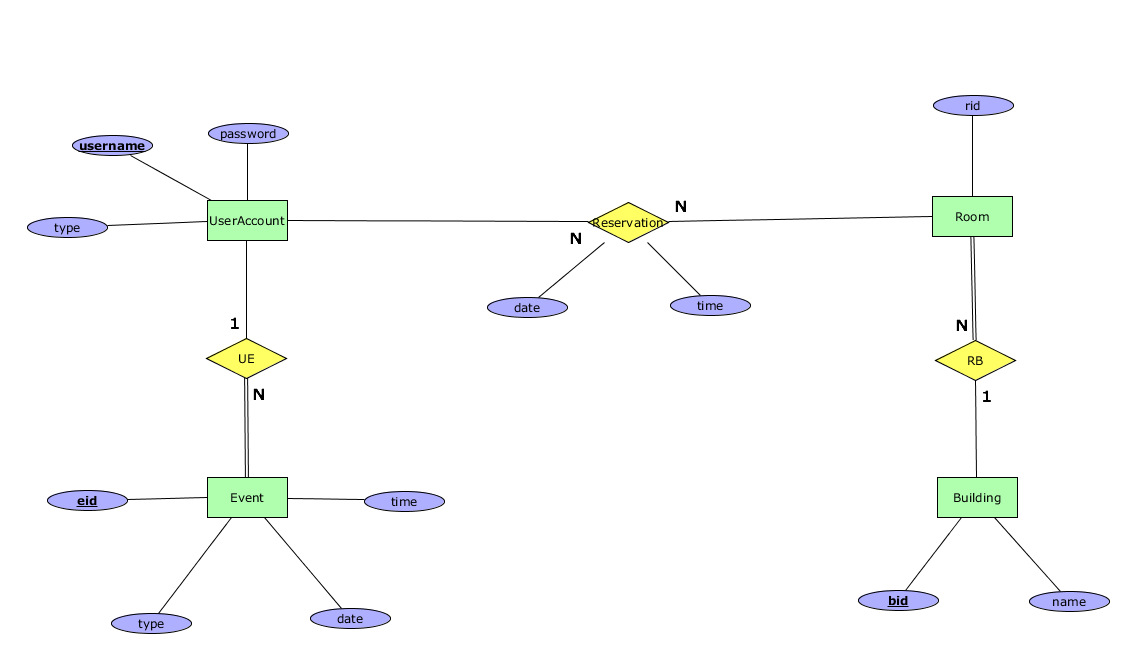


Figure 4.10 ER Diagram

**RELATIONAL MODEL**

BUILDING (bid, name)

EVENT (eid, time, date, type, usernameFK)

RESERVATION (accountFK, ridFK, date, time)

ROOM (rid*,* bidFK)

USERACCOUNT (username, password, type)

# SYSTEM DESIGN

## SUBSYSTEM DECOMPOSITION

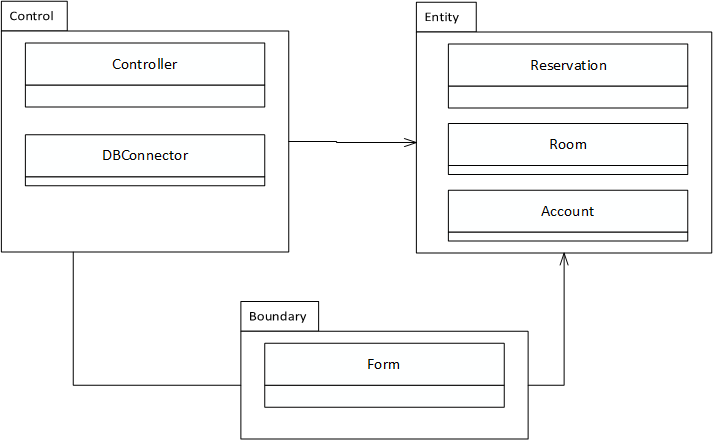


Figure 5.1 subsystems

# APPENDIX

## APPENDIX A-SOURCE CODE

namespace Entities

{

public class Account

{

private string username { get; set; }

private string password { get; set; }

private string role { get; set; }

private string name { get; set; }

public Account(string username, string role, string password, string name)

{

this.username = username;

this.role = role;

this.password = password;

this.name = name;

}

}

}

namespace Entities

{

public class Reservation

{

private int rid { get; set; }

private string usn { get; set; }

private int roomID { get; set; }

private DateTime dtg { get; set; }

public Reservation(string usn, int roomID, string datetime)

{

this.usn = usn;

this.roomID = roomID;

DateTime dt;

if (DateTime.TryParse(datetime, out dt) == true)

{

this.dtg = dt;

}

}

public Reservation(int rid, string usn, int roomID, DateTime dtg)

{

this.rid = rid;

this.usn = usn;

this.roomID=roomID;

this.dtg = dtg;

}

}

}

namespace Entities

{

public class Room

{

private int roomID { get; set; }

private string building { get; set; }

public Room(int roomID, string building)

{

this.roomID = roomID;

this.building = building;

}

}

}