**Requirements Analysis Document**

BLUE BOX System

CSCI 4711 Software Engineering

Fall 2016

Augusta University

Augusta, GA

Date: 10/10/2016

Version 2

Team Members

Will Patten

Michael Archer

Miguel Morell

Andrew Keller **Abstract**

This document contains the requirements, analysis and the design artifacts for the BLUE BOX CO. software system. Blue Box Co. is a media rental system that allows users such as customers to interact with the system to rent or return media at the kiosk. Verified administrators may also interact with the system by checking which rentals are available at the time.

The rest of this document is structured as follows. Chapter 1 contains the introduction. This chapter gives a brief description of the system and the scope of the system. Chapter 2 defines the functional requirements of the system. This chapter also goes over the use case of the system, the use case descriptions, and the sequence diagrams. Chapter 3 illustrates the user interface mockups for each form within the program. This chapter show the basic functionality and layout of the program. Chapter 4 contains Object design. These are class diagrams showing important attributes and operations, without attributes and operations, and their relationship with each other. Chapter 4 will also include the state chart diagrams for each controller. Chapter 5 will detail the Subsystem Decomposition. Chapter 6 will include source code for the entities of the system.**Table of Contents**

1 INTRODUCTION……………………………………………………………………..4

* 1. SCOPE OF THE SYSTEM…………….……………………………4
  2. OVERVIEW OF THE DOCUMENT…………………………...…..4

1. REQUIREMENTS OF SYSTEM…….…………………………………………..….5
   1. FUNCTIONAL REQUIREMENTS………………….….…………..5
   2. USE CASES……………………………………….…….………......6
   3. USE CASE DESCRIPTION……………………….…….…………..7
   4. REQUIREMENT ANALYSIS………………………………….......13
2. USER INTERFACE MOCKUPS……………………………………...……….......19
   1. RENT……………………………………………………….…….....19
   2. RETURN……………………………….…………………………...21
   3. LOGIN……………………………...……………………………….22
   4. LOGOUT……………………….…………………………………...23
3. OBJECT DESIGN………………………………………………………………….24
   1. OBJECT INTERACTION…………………………………..……….24
   2. DETAILED CLASS DESIGN………………………………………26
   3. STATECHART DIAGRAM………………………………………...28
4. SYSTEM DESIGN……………………………………………………...…………..32
   1. SUBSYSTEM DECOMPOSTITION……………………………......32
5. APPENDIX……………………………………………………………………........33
   1. APPENDIX A – SOURCE CODE…………………………………..33
6. **INTRODUCTION**

**1.1 SCOPE OF SYSTEM**

BlueBox is a rental system for consumer media. It includes two actors, a *Customer* that represents a consumer that can rent and return particular media from the system and the *Administrator* that can log in and view the database of rented and non-rented media then log out.

The system supports both actors by keeping track of availability of the rentals. The software has access to a database which shows whether customers are able to rent media based on availability. The Customer and the Administrator both access the system using a stand-alone kiosk.

The functionality for renting, returning, logging in, and logging out is completely within the scope of this system.

**1.2 OVERVIEW OF DOCUMENT**  
Chapter 2 Outlines the functional requirements of the system. Within this chapter is a list of functional requirements of BlueBox. It also includes a use case model of those functional requirements. A detailed description of each functional requirement then follows. Chapter 3 shows the user interface mockups. Chapter 4 detail class diagrams, their relationship with each other, and the state chart diagrams. Chapter 5 shows the subsystem decomposition. Chapter 6 will show the appendix that includes source code for entities.

1. **REQUIREMENTS OF SYSTEM**

**2.1 FUNCTIONAL REQUIREMENTS**

* Rent – The customer invokes this function to choose from a list of available media to rent then it will allow the customer to enter their credit card information and add it to the database.
* Return – The customer invokes the kiosk to return previously rented media. It will allow the customer to enter their credit card information and media ID number. Once returned, it will take their information off of the database and set the media back to being available to be rented again.
* Login – The administrator invokes this function and the system will allow the administrator to enter their credentials. Once entered, the system will then allow access to the entire media database and the database of media rented by which customers.
* Logout – Once Logged in, the system will give access to logout, if invoked it will prompt the system to the Main Menu and disallow access to the databases.

**2.2 USE CASES**

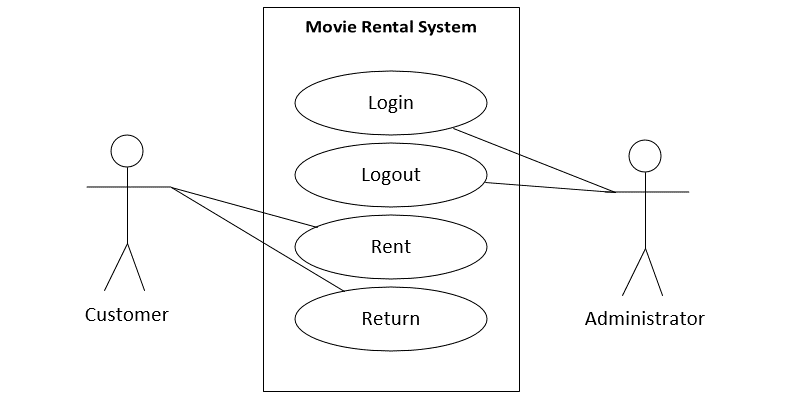


Figure 2.1: Use Case Diagram for Movie Rental System

**2.3 USE CASE DESCRIPTIONS**

*Use case name*: Rent

*Participating actors*: Customer

*Flow of events*: 1. The customer selects the Rent button on the Main Menu.

**2. Kiosk presents the movie form with a list of all currently available movies.**

3. The customer will select the desired rental for the displayed selectable list and presses the submit button.

**4. Kiosk presents the confirm rental form for the user to enter their credit card information and make the purchase.**

5. The user enters their credit card information and submits the form.

**6. Kiosk verifies that the credit card information is valid, and removes the purchased rentals from the available rentals list. Kiosk displays the Main Menu.**

*Entry condition*: Customer has a valid credit card.

*Exit condition*: Kiosk dispenses the movie and returns to the Main Menu.

*Security requirements*: The credit card information is typed as asterisks preventing on- lookers from seeing their private information.  
SQL queries for this use case are generated in a manner that prevents SQL injection attacks.

Figure 2.2: Rent

*Use case name*: Return

*Participating actors*: Customer

*Flow of events*: 1. The customer selects the Return button on the Main Menu.

**2. Kiosk presents the return form for the user to enter their credit card number and rental ID.**

3. The customer enters the rental ID, and their credit card information and presses the confirm button.

**4. Kiosk makes the returned rental available again in the list of available rentals. Kiosk displays Main Menu.**

*Entry condition*: The user has a valid credit card, and rental ID

*Exit condition*: Kiosk receives the rented movie and returns to the Main Menu.

*Security requirements*: All SQL queries for this use case will be generated in a manner that prevents SQL injection attacks.

Figure 2.3: Return

*Use case name*: Return (Failure)

*Participating actors*: Customer

*Flow of events*: 1. The customer selects the Return button on the Main Menu.

**2. Kiosk presents the return form for the user to enter their credit card number and rental ID.**

3. The customer enters the rental ID and presses the confirm button.

**4. The Kiosk displays an error, because the MovieID was invalid,**

*Entry condition*: The user enters the rental ID

*Exit condition*: Kiosk receives the rented movie and returns to the Main Menu.

*Security requirements*: All SQL queries for this use case will be generated in a manner that prevents SQL injection attacks.

Figure 2.3: Return (Failure)

*Use case name*: Login (Success)

*Participating actors*: Administrator

*Flow of events*: 1. The administrator selects the login button on the Main Menu.

**2. Kiosk presents the login form for the administrator to enter their username and password.**

3. The administrator enters their username and password in the form and selects the login button.

**4. Kiosk presents the Administrative Menu form with a list of available and unavailable rentals.**

*Entry condition*:

*Exit condition*: The Main Menu is displayed to the administrator.

*Security requirements*: Password will be displayed with asterisks instead of plain text. All SQL queries for this use case will be generated in a manner that prevents SQL injection attacks.

Figure 2.4: Login (Success)

*Use case name*: Login (Failure)

*Participating actors*: Administrator

*Flow of events*: 1. The administrator selects the login button on the Main Menu.

**2. Kiosk presents the login form for the administrator to enter their username and password.**

3. The administrator enters their username and password in the form and selects the login button.

**4. Because the login credentials are invalid, the Kiosk presents an error message and opens the main menu.**

*Entry condition*:

*Exit condition*: The Main Menu is displayed to the administrator.

*Security requirements*: Password will be displayed with asterisks instead of plain text. All SQL queries for this use case will be generated in a manner that prevents SQL injection attacks.

Figure 2.4: Login (Failure)

*Use case name*: Logout

*Participating actors*: Administrator

*Flow of events*: 1. The administrator selects the logout button located on the Administrative Menu form.

**2. Kiosk returns to the Main Menu.**

*Entry condition*: The administrator is logged in.

*Exit condition*: The administrator is logged out and the system returns to the Main Menu.

*Security requirements*:

Figure 2.5: Logout

**2.4 SEQUENCE DIAGRAMS**

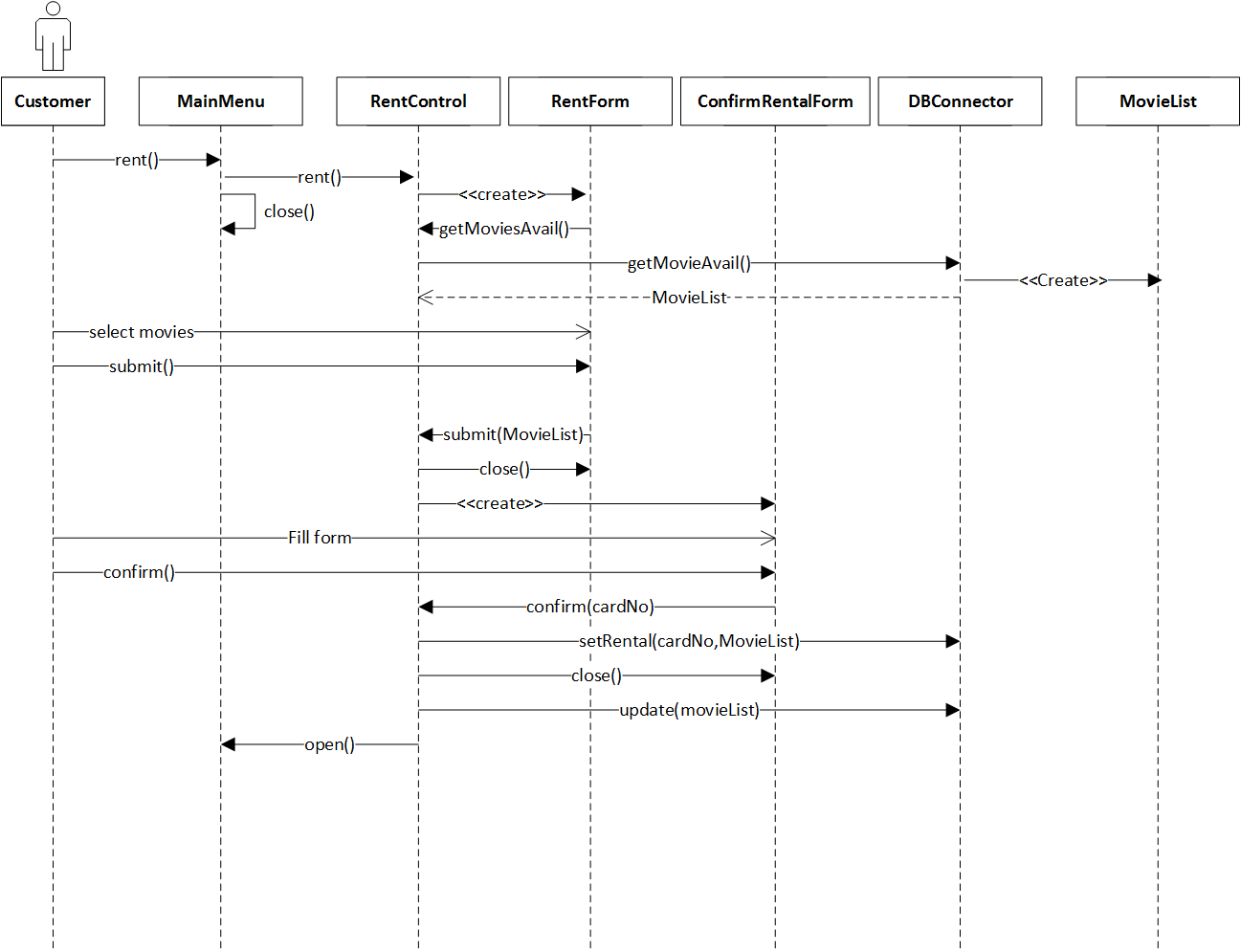


Figure 2.4.1: Rent Sequence Diagram

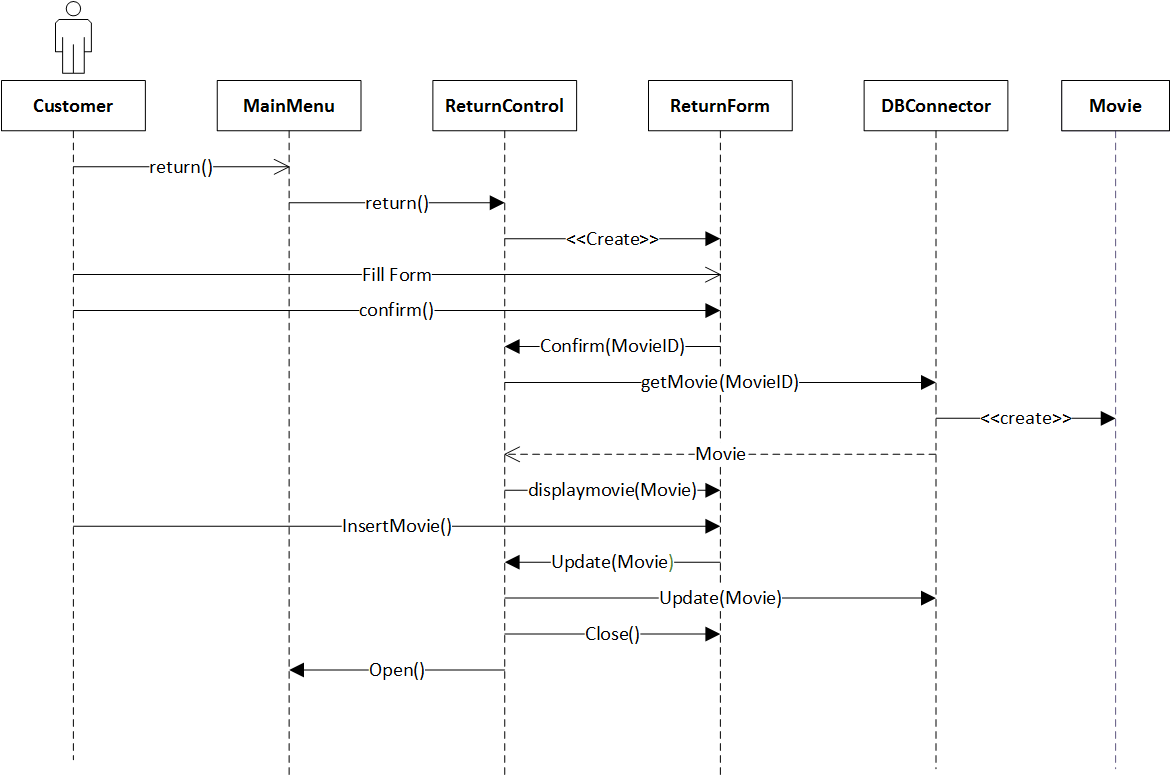


Figure 2.4.2: Return Sequence Diagram

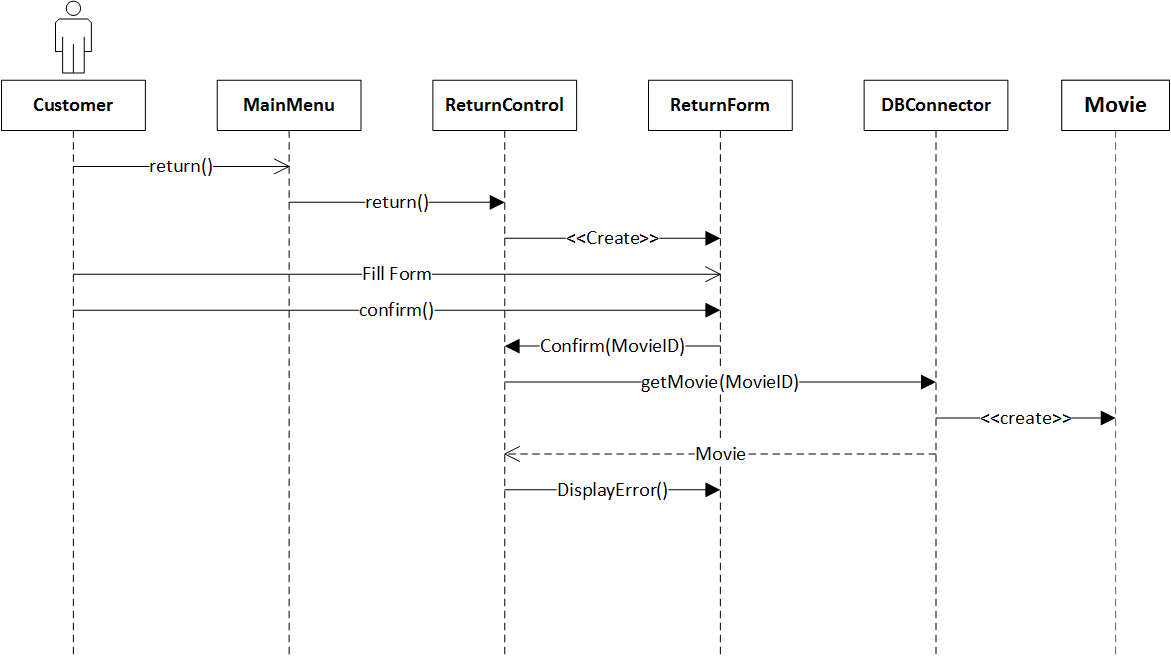


Figure 2.4.3: Return Sequence Diagram (Failure)

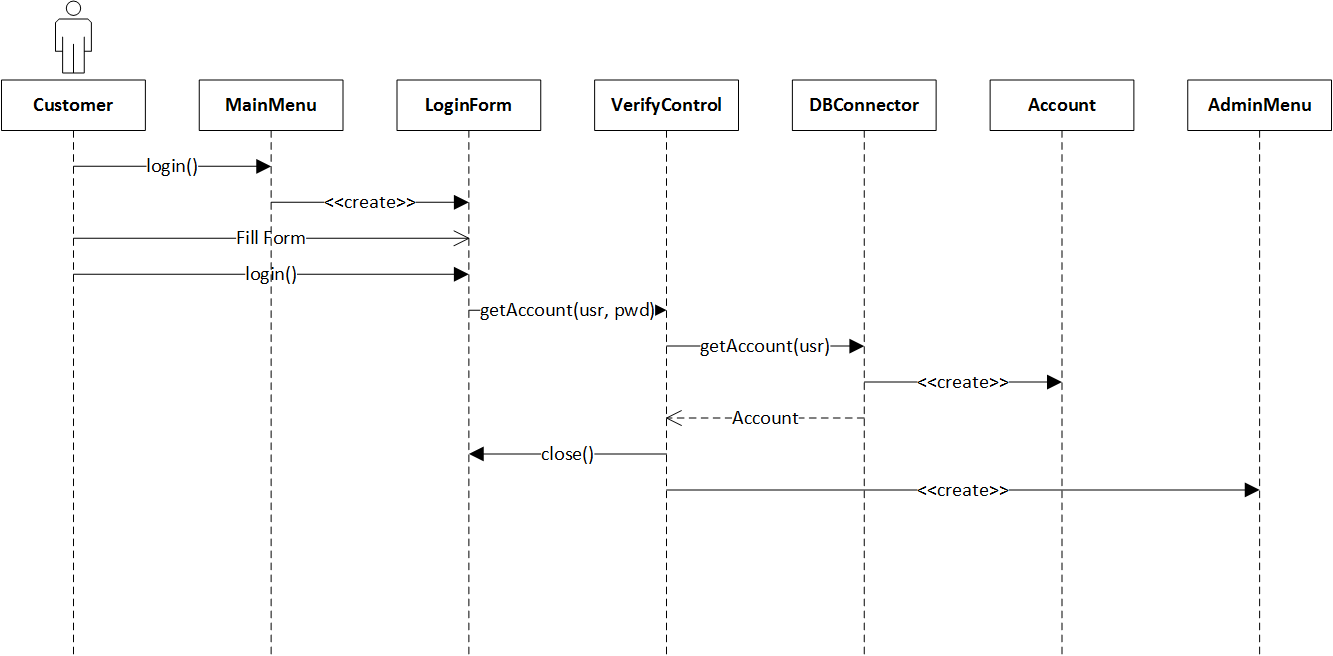


Figure 2.4.4: Login (Success) Sequence Diagram

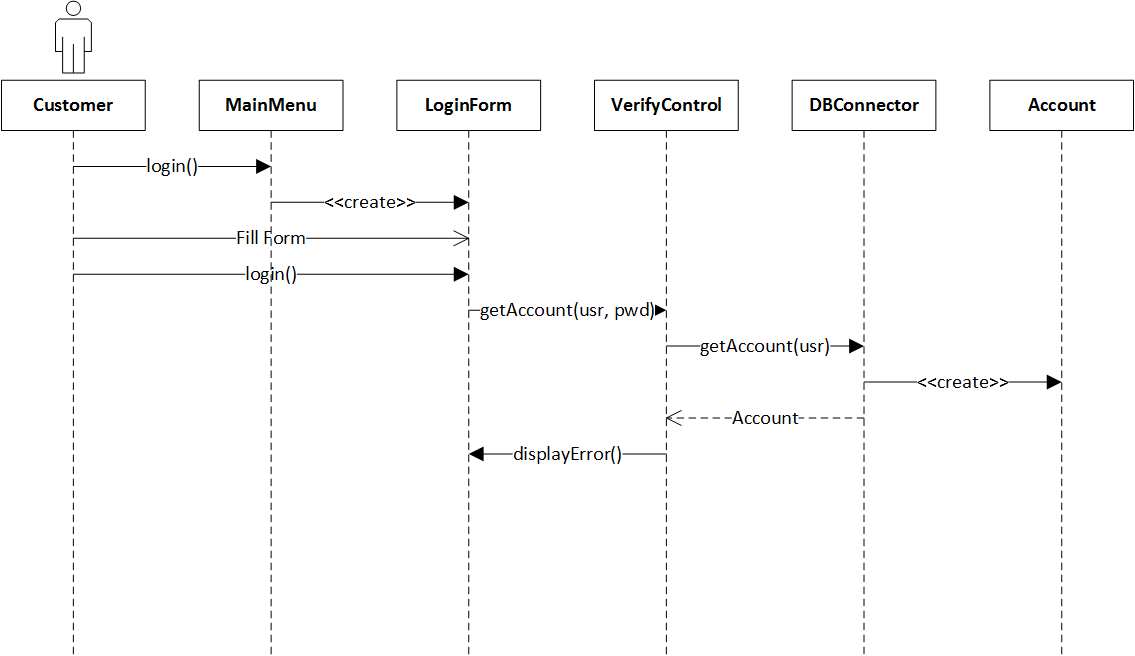


Figure 2.4.5 Login (Failure) Sequence Diagram

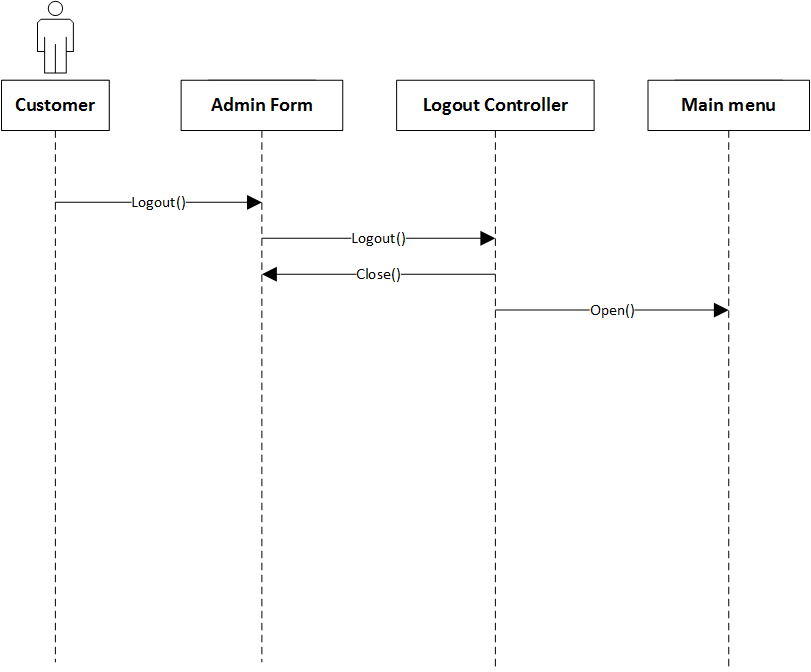


Figure 2.4.6: Logout Sequence Diagram

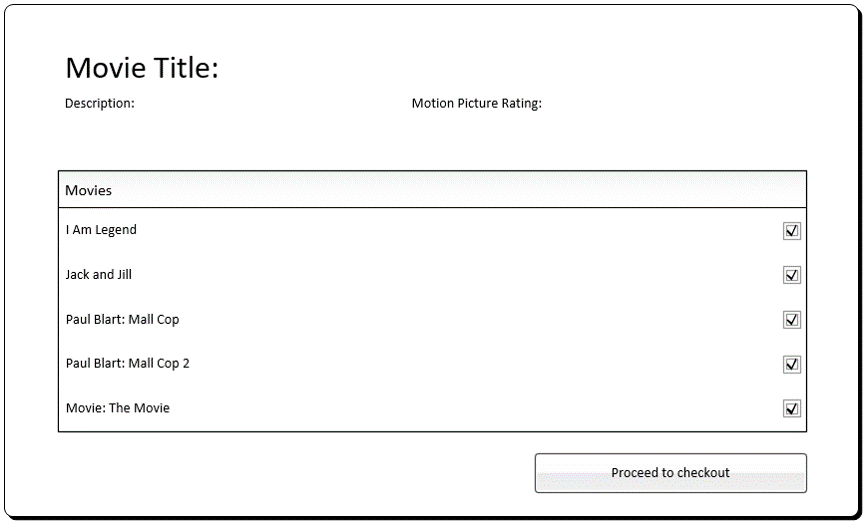
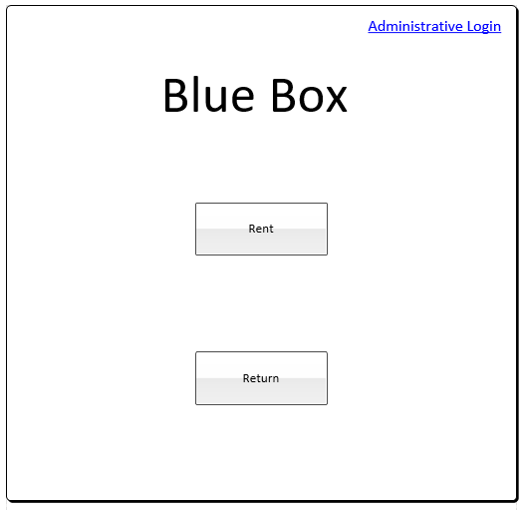
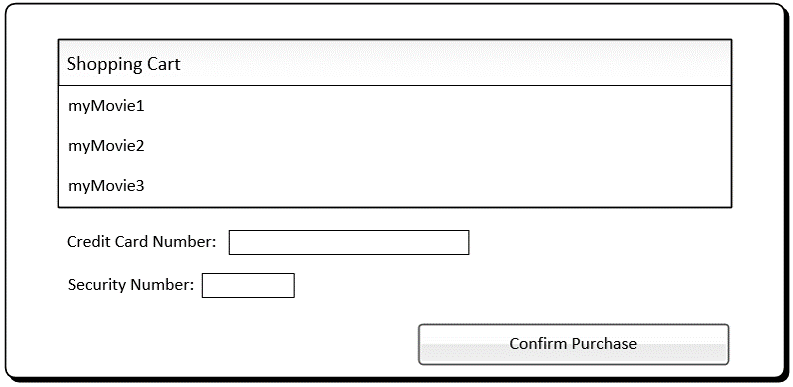
1. **USER INTERFACE MOCKUPS  
    3.1 RENT**  
   

Figure 3.1: Main Menu

Figure 3.2: Available Rentals Menu

Figure 3.3: Confirm Purchase Menu

**3.2 RETURN**

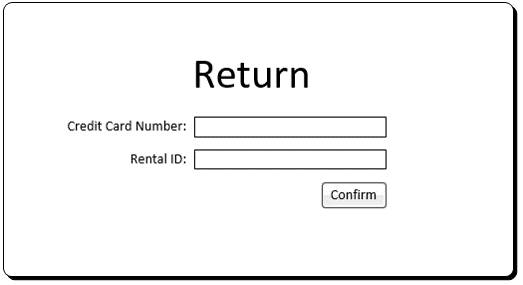


Figure 3.4: Return Menu

* 1. **LOGIN**

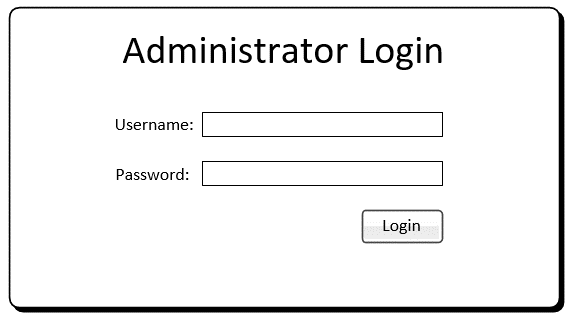


Figure 3.5: Administrator Login Menu

**3.4 LOGOUT**

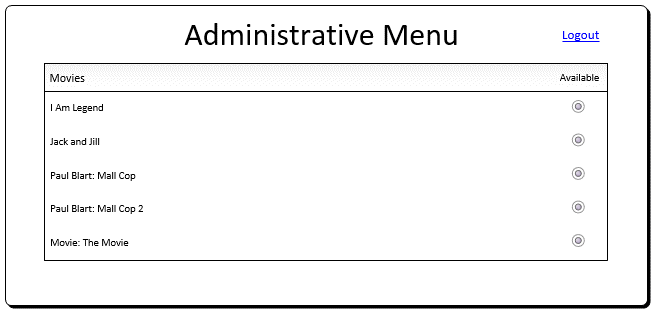


Figure 3.6: Administrative Logout Menu

1. **OBJECT DESIGN**

**4.1 OBJECT INTERACTION**

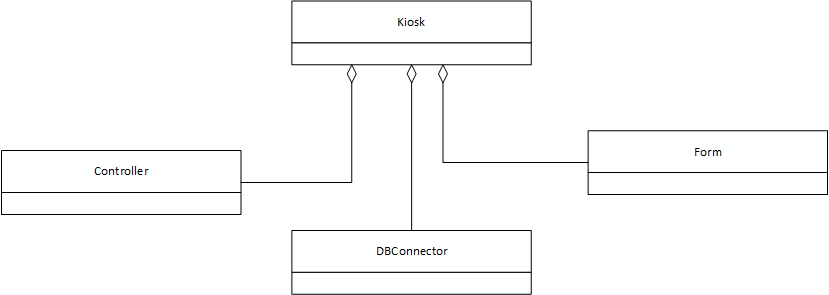


Figure 4.1: Class Diagram

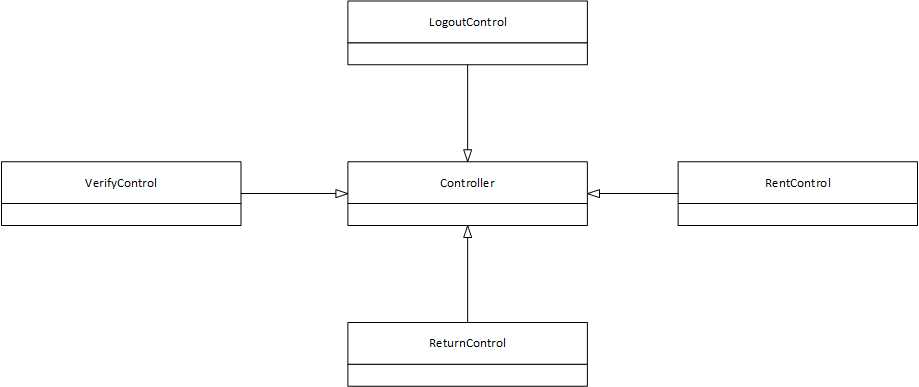


Figure 4.2: Class Diagram: Controller

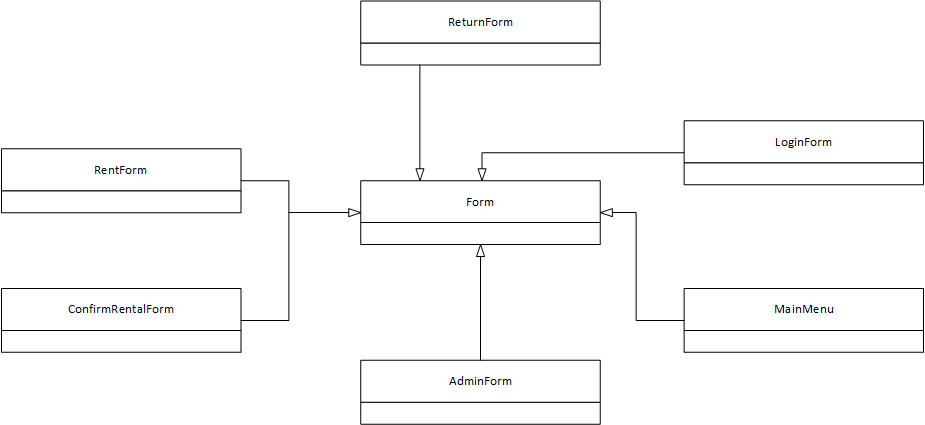


Figure 4.3 Class Diagram: Form

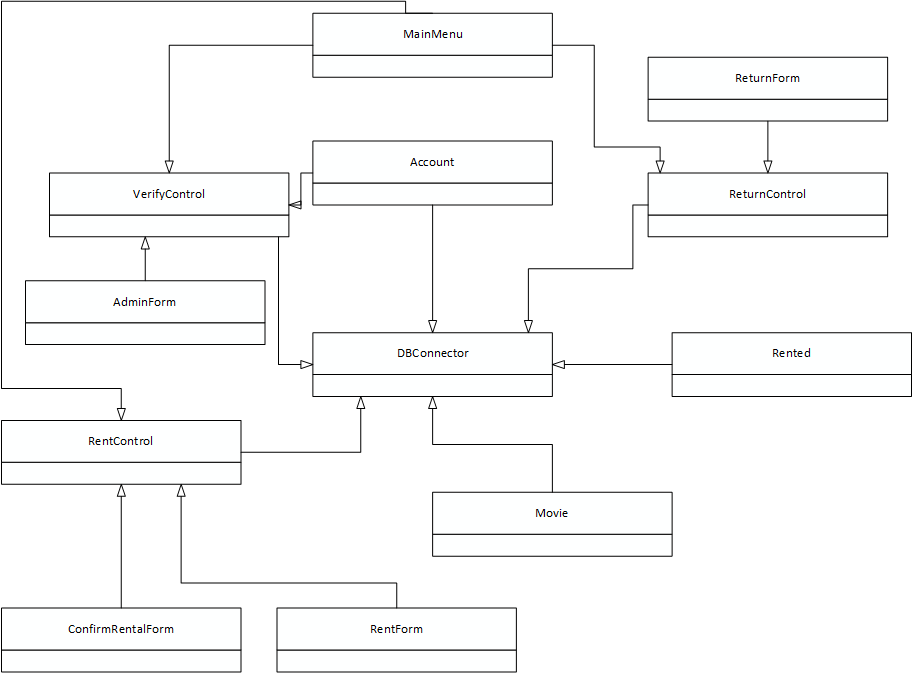


Figure 4.4: Class Diagram

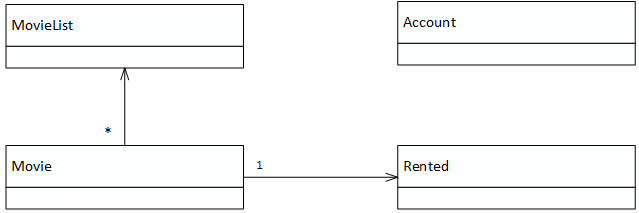
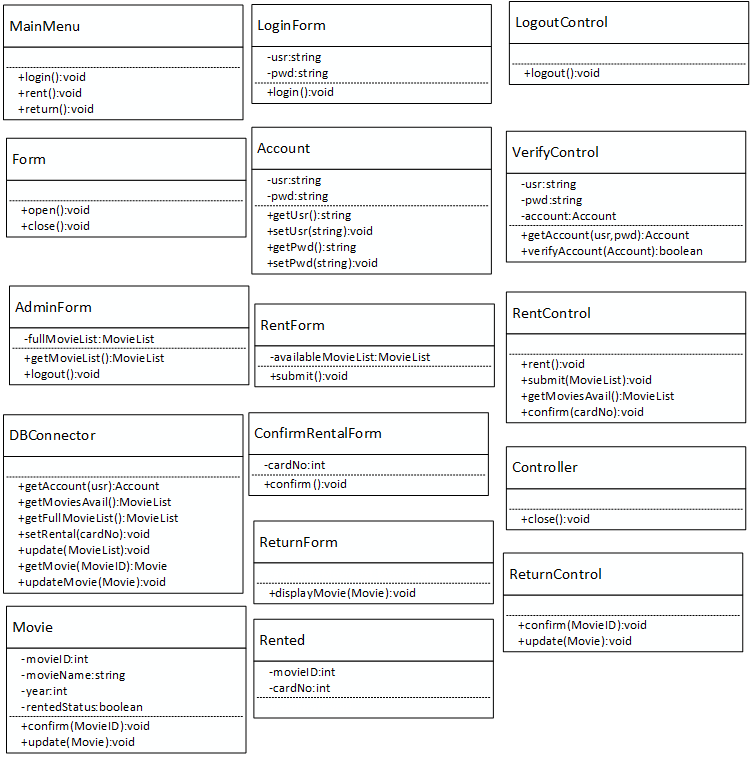


Figure 4.5: Class Diagram Entity

**4.2 DETAIL CLASS DESIGN**



**4.3 STATECHART DIAGRAM**

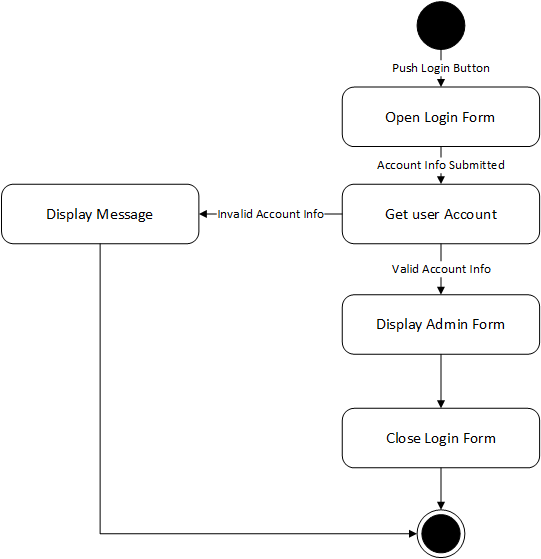


Figure 4.3.1: Login State Chart Diagram

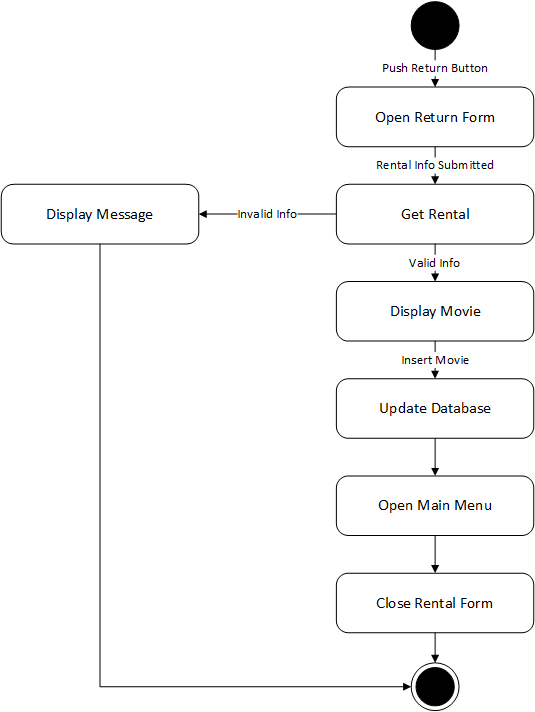


Figure 4.3.2: Rental State Chart Diagram

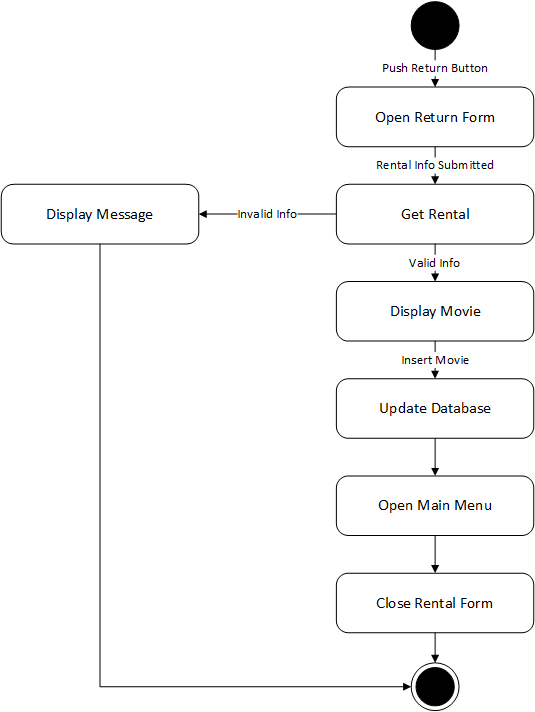


Figure 4.3.3: Return State chart Diagram

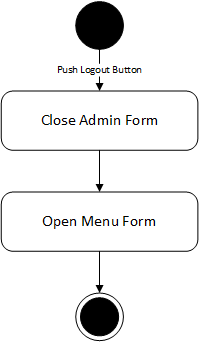


Figure 4.3.4: Logout State Chart Diagram

1. **SYSTEM DESIGN**
   1. **SUBSYSTEM DECOMPOSTITION**

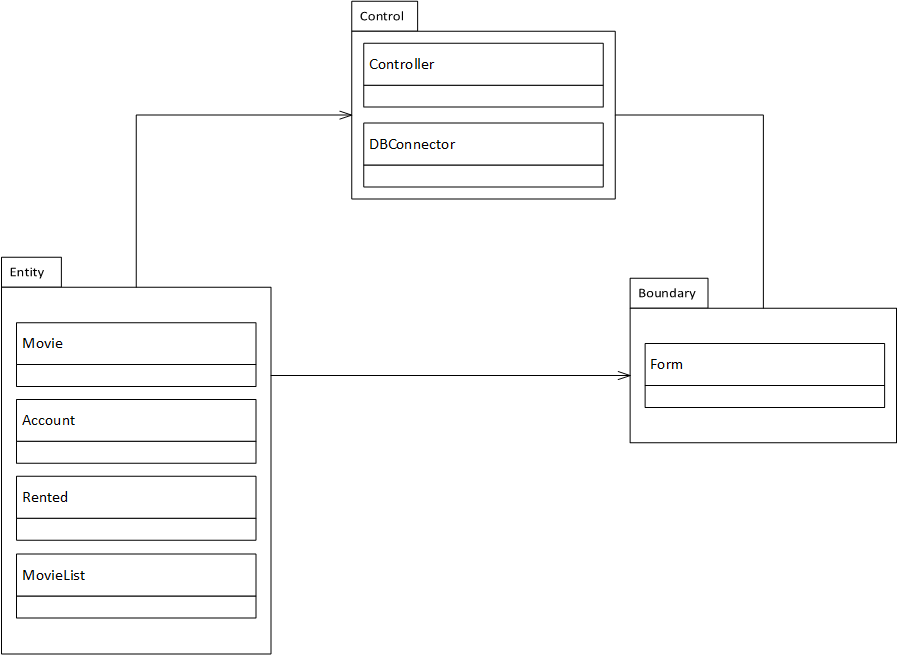


Figure 5.1: Subsystem Decomposition Diagram

1. **APPENDIX**
   1. APPENDIX A – SOURCE CODE

public class Movie

    {

        private int id;

        private string movieName;

        private int year;

        private bool rentedStatus;

        public int Id

        {

            get

            {

                return id;

            }

            set

            {

                id = value;

            }

        }

        public string MovieName

        {

            get

            {

                return movieName;

            }

            set

            {

                movieName = value;

            }

        }

        public int Year

        {

            get

            {

                return year;

            }

            set

            {

                year = value;

            }

        }

        public bool RentedStatus

        {

            get

            {

                return rentedStatus;

            }

            set

            {

                rentedStatus = value;

            }

        }

        public void Confirm(int movieId)

        {

        }

        public void Update(Movie movie)

        {

        }

        public Movie()

        {

        }

    }

    public class Account

    {

        private string usr { get; set; }

        private string pwd { get; set; }

        public string Usr

        {

            get

            {

                return usr;

            }

            set

            {

                usr = value;

            }

        }

        public string Pwd

        {

            get

            {

                return pwd;

            }

            set

            {

                pwd = value;

            }

        }

        public Account()

       {

        }

    }