**C868 – Software Capstone Project Summary**

**Task 2 – Section C**

|  |  |
| --- | --- |
| **Capstone Proposal Project Name:** | http://www.idevnews.com/views/images/uploads/general/wgu_logo.png  Movie Rental Web Application |
| **Student Name:** | Ganeshkumar Miriyala |

Table of Contents

[Application Design and Testing 3](#_Toc530558777)

[Design Document 3](#_Toc530558778)

[Class Design 3](#_Toc530558779)

[UI Design 3](#_Toc530558780)

[Unit Test Plan 4](#_Toc530558781)

[Introduction 4](#_Toc530558782)

[Purpose 4](#_Toc530558783)

[Overview 4](#_Toc530558784)

[Test Plan 4](#_Toc530558785)

[Items 4](#_Toc530558786)

[Features 4](#_Toc530558787)

[Deliverables 5](#_Toc530558788)

[Tasks 5](#_Toc530558789)

[Needs 5](#_Toc530558790)

[Pass/Fail Criteria 5](#_Toc530558791)

[Specifications 5](#_Toc530558792)

[Procedures 6](#_Toc530558793)

[Results 6](#_Toc530558794)

[C4. Source Code 6](#_Toc530558795)

[C5. Link to Live Version 7](#_Toc530558796)

[User Guide 8](#_Toc530558797)

[Introduction 8](#_Toc530558798)

[Installation and Using the Application 8](#_Toc530558799)

[*Main Page* 9](#_Toc530558800)

[*Login and Signup* 9](#_Toc530558800)

[*Movie page* 9](#_Toc530558801)

[*Accesss Movie List and Rent* 9](#_Toc530558802)

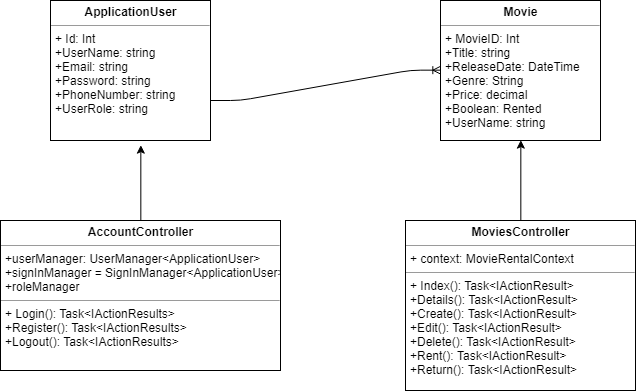
[*Reports* 10](#_Toc530558803)

# Application Design and Testing

# Design Document

## Class Design

The design is the basic architecture of the class design for the application. The application follows an MVC (Model-View-Controller) architecture using ASP.NET CORE 2.0. The Application User is the class that keeps track of the application user whether it be an “Admin” or a Customer. The list of movies will be shown to all users and each user can rent multiple movies. The MVC architecture lays out the following application where UI presentation logic is in the View; the business logic is defined in the controller classes and the data to be stored is defined in the model classes. Data from the UI is moved through the View to the controller where it is to be processed. The processed data flows into the model where process data storage is defined before moving to the database. The reverse process of data moving from the database to the UI is similar to the process of data moving from the UI to the database.



## UI Design

The two UI design elements provided are a low-fidelity and a high-fidelity design image of the Movie Rental main page. At the top of the images are the header where a user can register or login to the website before renting movies. Once the user is registered and logged in, they should be able to see the List of movies available along with the movie information and button links to movie details, and an option to rent in a table format. Administrators will have extra options to edit movie details or delete movies from the list. Below the available movies will be another table list of the rented movies for the specific user. The overall design of the web application will use more whitespace background to allow users to key in on important content on the site and not feel overloaded with cluttered information.

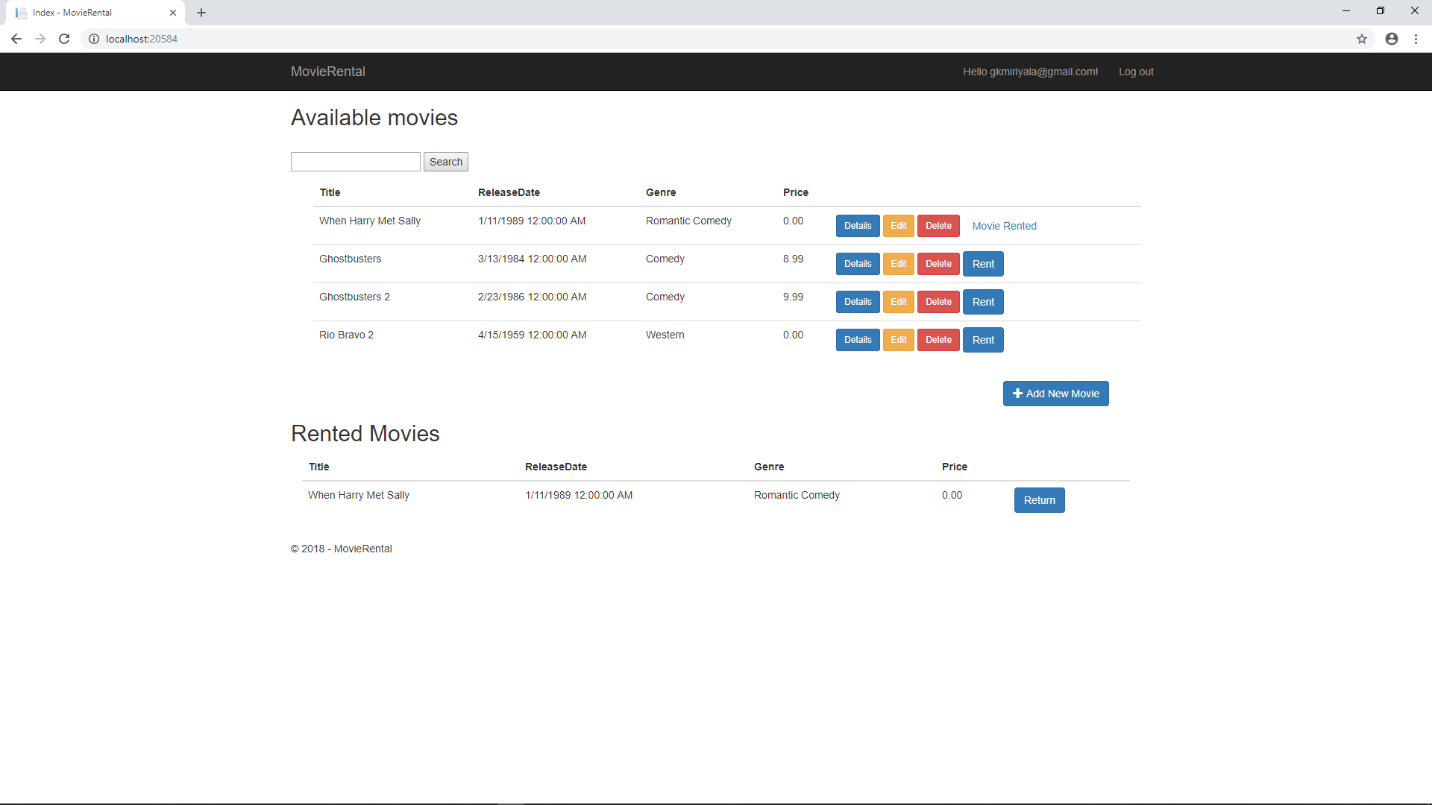
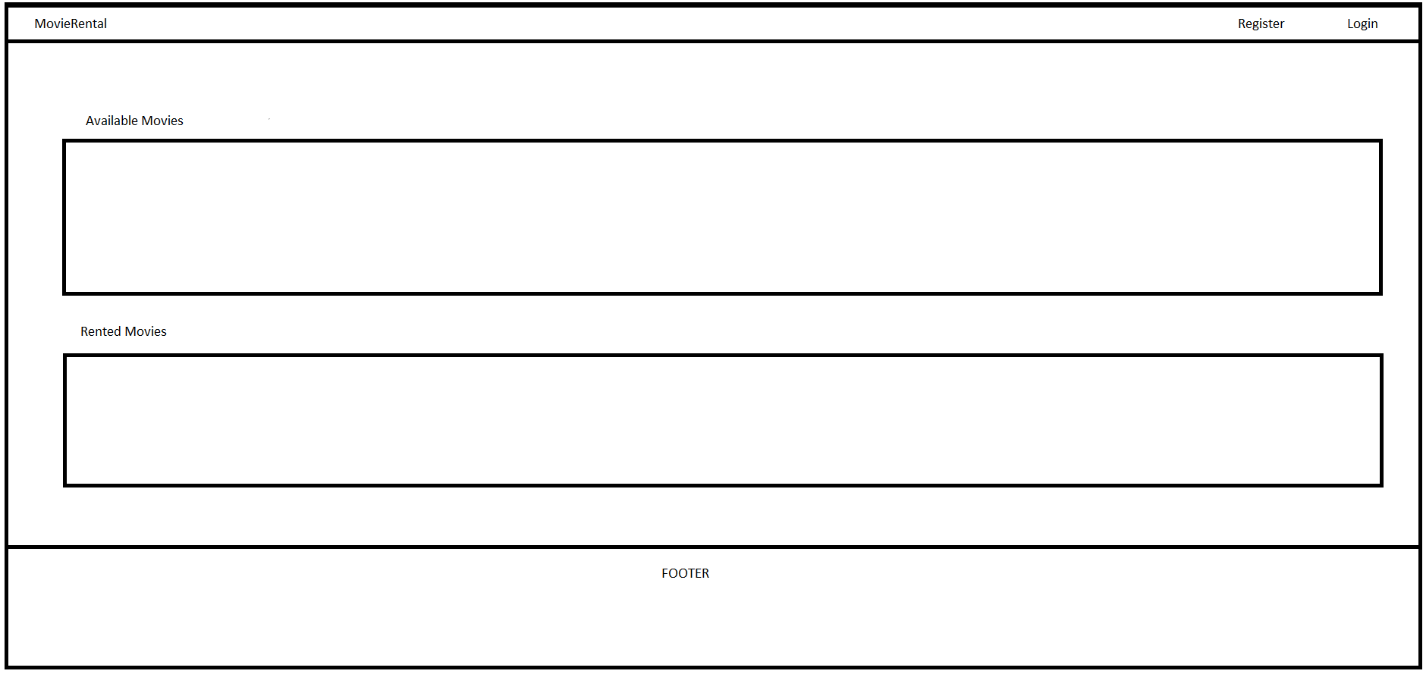


Figure 1: Low Fidelity

Figure 2: High Fidelity

# Unit Test Plan

## Introduction

### Purpose

The testing method that I used were unit tests on individual methods in the application. These tests revealed methods were yielding the appropriate response and which ones weren’t. This made it much quicker to find and fix the methods that had errors or were not working as intended. This testing method was quicker than having to run the entire application in order to test specific methods because not only could I test each one individually all in one run, but I could run these tests multiple times because these tests were automated and saved.

### Overview

The testing method that I used was the xUnit tests that are part of the ASP.NET CORE package in Visual Studio. I used these tests to individually test controller methods for my AccountController and MoviesController. The methods in the controllers are supposed to return the appropriate views that logically describe how the appropriate page is supposed to be presented in the UI. These tests revealed which controller methods were yielding the appropriate view page tied to the specific method and which ones weren’t. Some of the functions that were tested were the Index(), Create(), Edit, Delete() functions in the MoviesController. I tested whether the functions were returning the correct results, and processing data in an expected manner. For example, I tested the delete function by inputting the appropriate MovieID number in the unit test and having the unit test result check whether the movie was actually deleted or not. Similarly, I performed this test on the create and edit function for their respective results. I continued performing these test on all methods in all of my controllers until all the methods passed all the units tests.

## Test Plan

### Items

The following items are required to complete the unit tests

* Visual Studio 2017 IDE
* xUnit Testing Package
* Reference Packages
* Testing code File

### Features

The Functions/Features that are part of the test are all the methods from all controllers

AccountController – Login(), Register(), LogOut()

MoviesController – Index(), Details(), Create(), Edit(), Delete(), Rent(), Return()

### Deliverables

The test would show a pass/fail results in the test Explorer of the IDE environment (Visual Studio 2017). The detailed output would be provided in the console output which could then be copy and pasted to a notepad file for documentation purpose.

### Tasks

List the tasks required to complete the testing and provide the outcomes you identified.

The tasks required to complete the testing and provide the outcomes are as follows:

1. Download asp.net core xUnit Testing package
2. Setup the Testing Project using the xUnit package in the same solution as the main Project
3. Download the required dependencies and references
4. Write the unit test code in the testing file belonging to the testing project
5. Build the unit test code
6. Run the tests for a pass/fail result
7. Fix the methods that failed the test
8. Repeat step 6 and 7 until all tests pass

### Needs

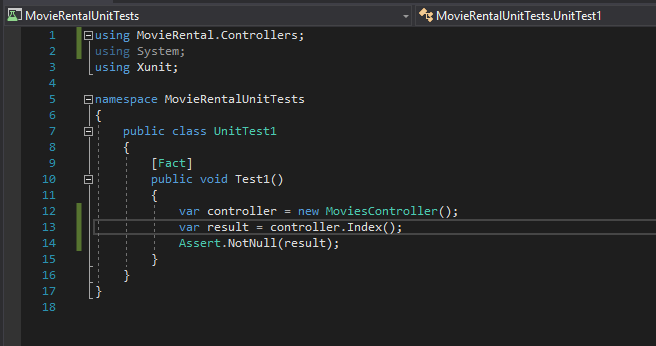
The following were needed to perform the tests:

1. Visual Studio 2017
2. Asp.NET core 2.0 sdk package
3. XUnit Test Project (.NET Core)
4. Written Unit Tests in Testing File
5. Test Explorer running

### Pass/Fail Criteria

The results of the test were automated through the testing package. For each unit test there were three pieces of code. The first was to declare the controller I was testing, the second was to declare the method in the controller that I was testing and third was to declare the result to be testing. Once the unit tests were written for all aspect of each controller method I wanted to test, I ran the test in the test explorer of the Visual Studio IDE. The test explorer displayed the pass/fail results for each unit test. The criteria were to get a pass for all unit tests in order to declare completion. If the test failed, I had to go back and rework on the failed portions of each failed method until each method passed all of its unit tests.

## Specifications

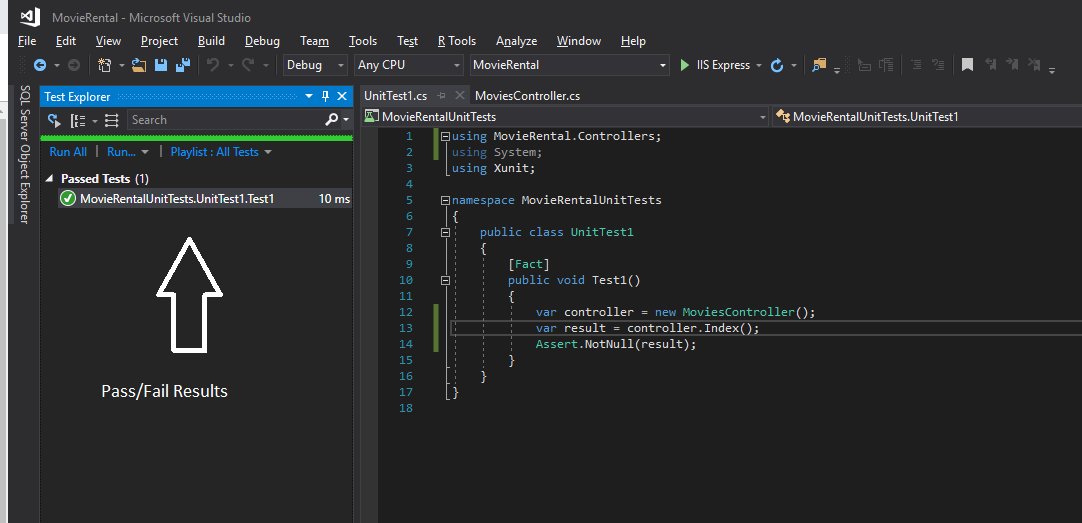


## Procedures

The steps to complete the testing process were as follows:

1. Write Acceptance Unit Tests for the methods to be created
2. Write the code for the methods to be tested
3. Run the Unit Test Application
4. Check the tests for the methods that failed the unit test
5. Fix the errors for the methods that failed
6. Rerun all the unit tests
7. Repeat step 4 – 6 until all tests pass

## Results



# C4. Source Code

gkm\_C868\_MovieRental.zip

# C5. Link to Live Version

<http://wgumovierentalappfinal.azurewebsites.net/>

# User Guide

## Introduction

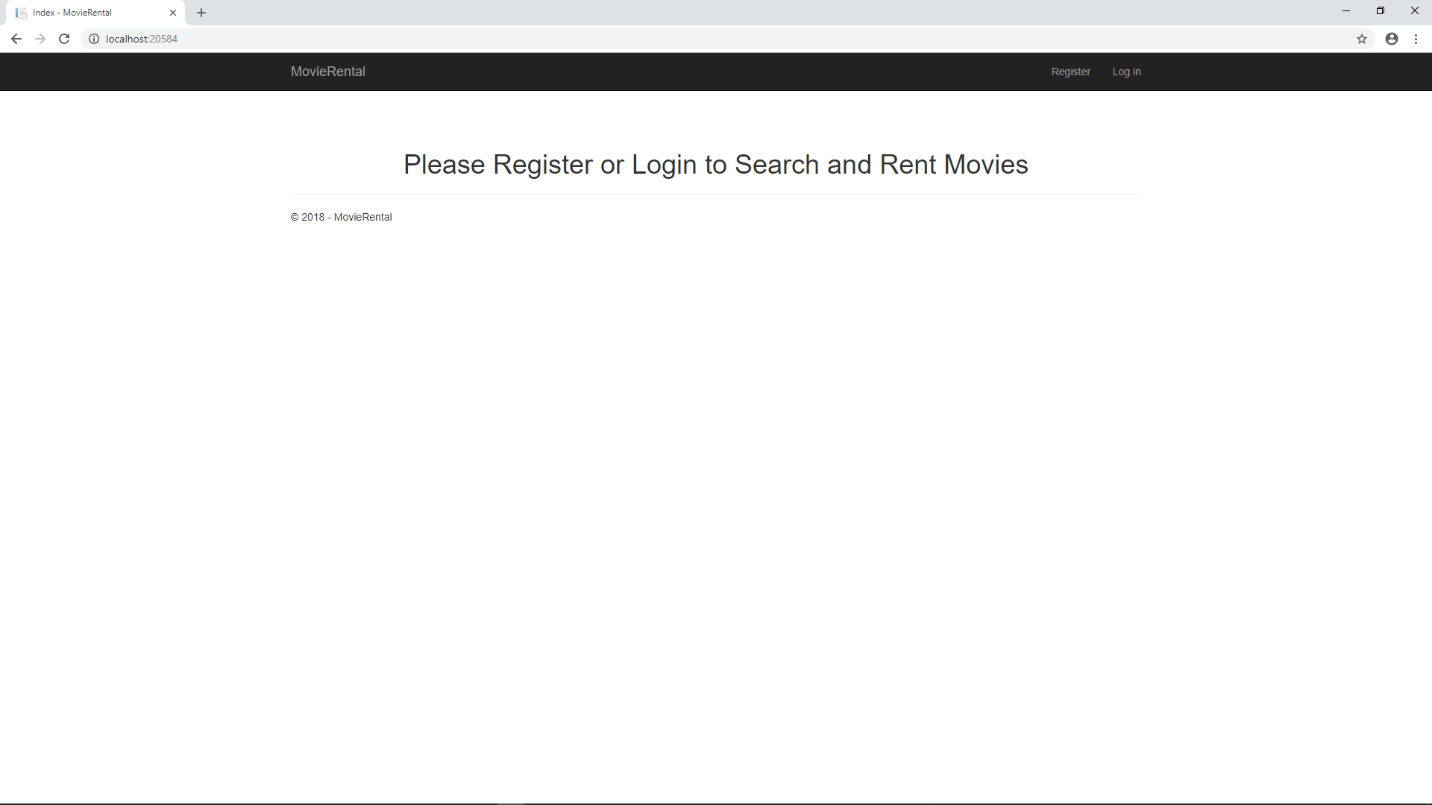
A fully functional Asp.net Movie Rental Web Application and SQL server database stored on a cloud server is being provided. The user will access the application using a supported internet browser (google chrome, Mozilla FireFox, Microsoft Edge) through the website URL provided.

## Installation and Using the Application

The Web Application will be fully hosted on the cloud server and will be available to access through a provided link by all user using an internet browser of their choice. The following will guide users in using the application.

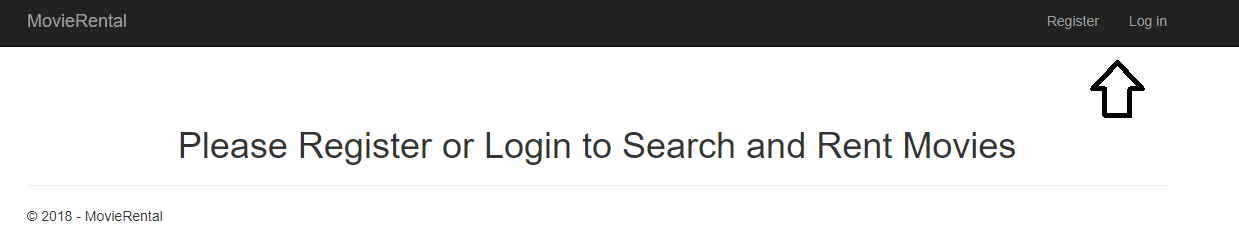
## *Main Page*

The main page will request the user to register or login to the application before preceding

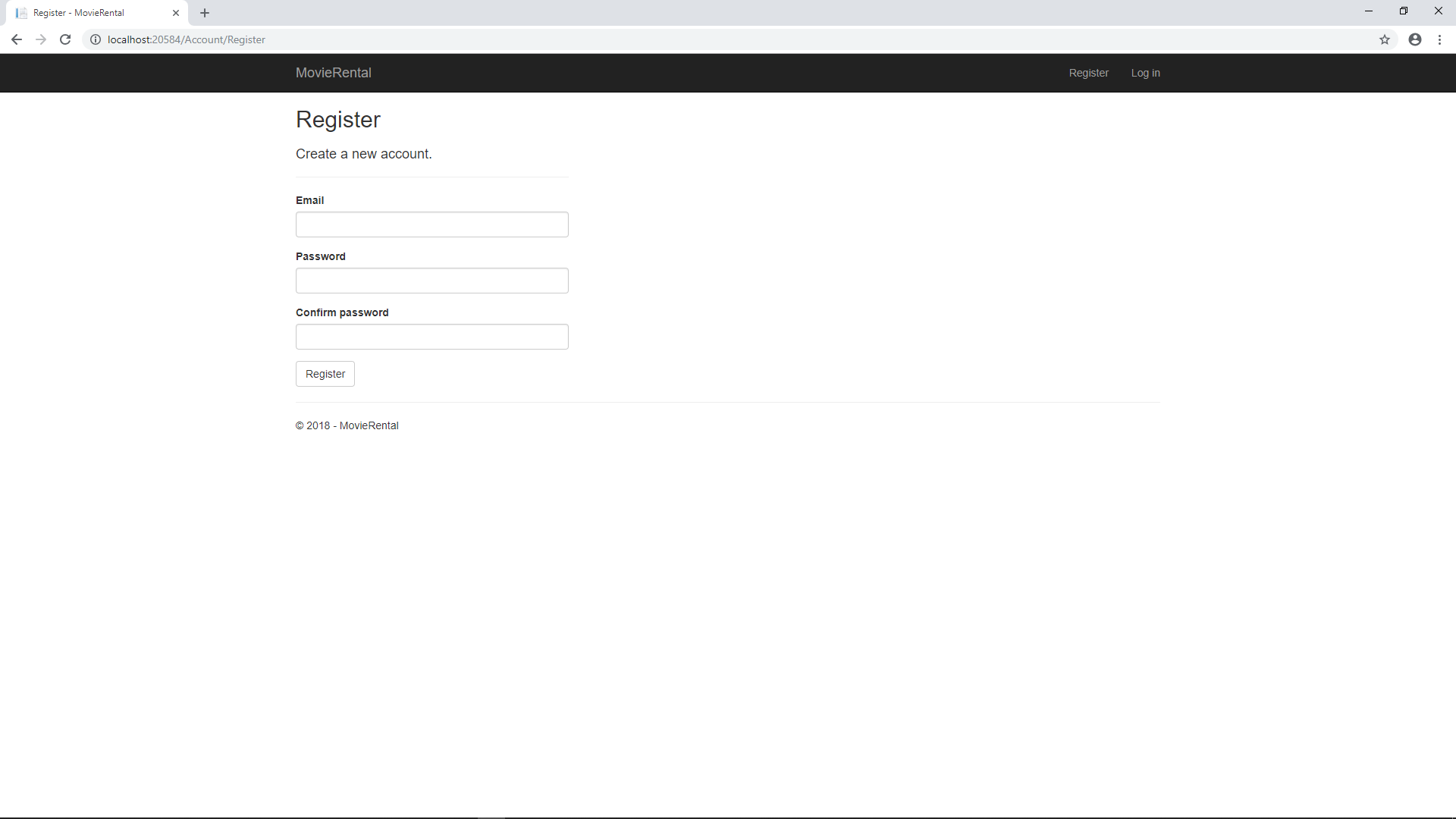


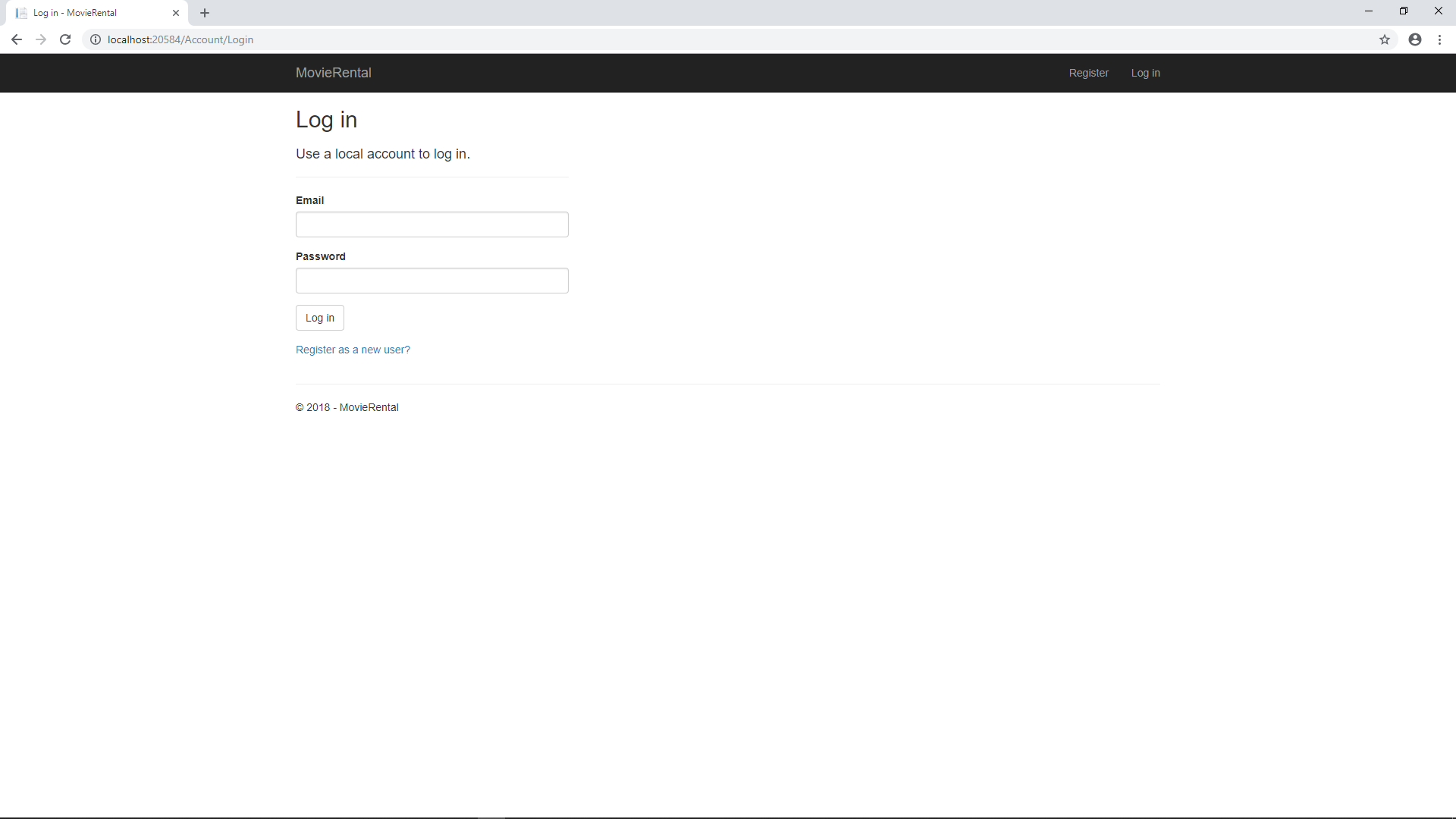
## *Login and Signup*

1. *Click the “Register” for new users or "Log in" for registered users in the top right corner of the app.*



1. *The “Register” button will lead to the Registration page and the Login button will lead to the “Log In” page.*

*Register Page*

*Login Page*

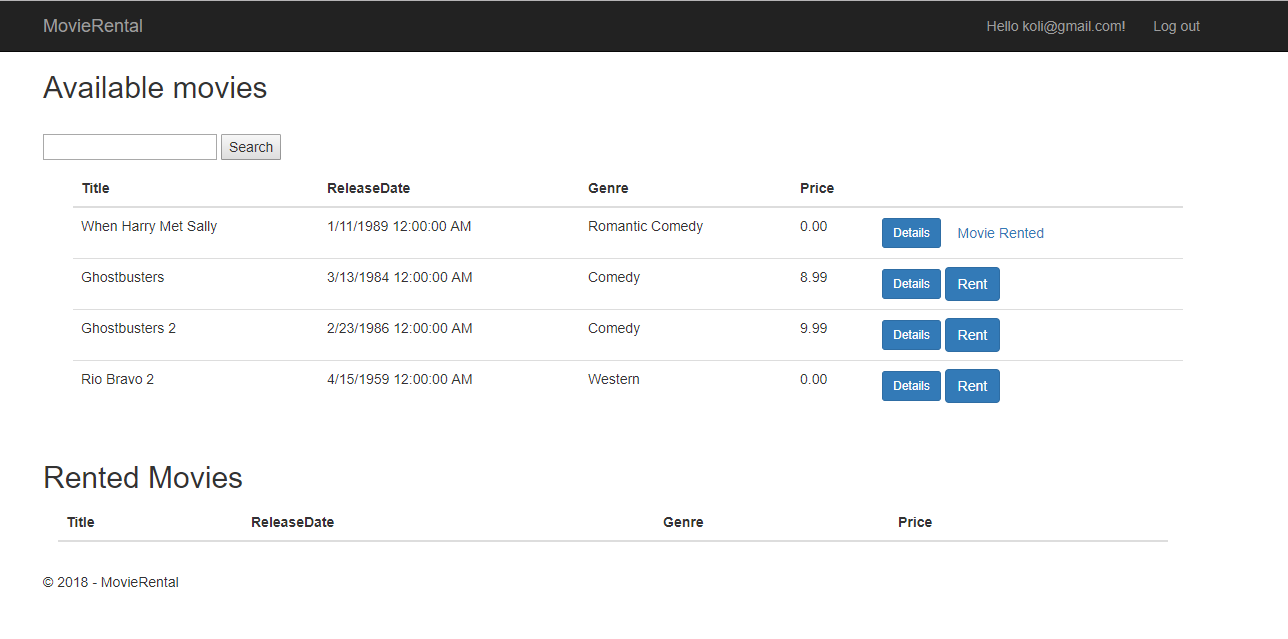
1. *If you need to create an account, choose a unique username and password. By default, the password requires at least 8 characters and a special charaction. This function could be changed to address new password requirements.*

## *Movie Page*

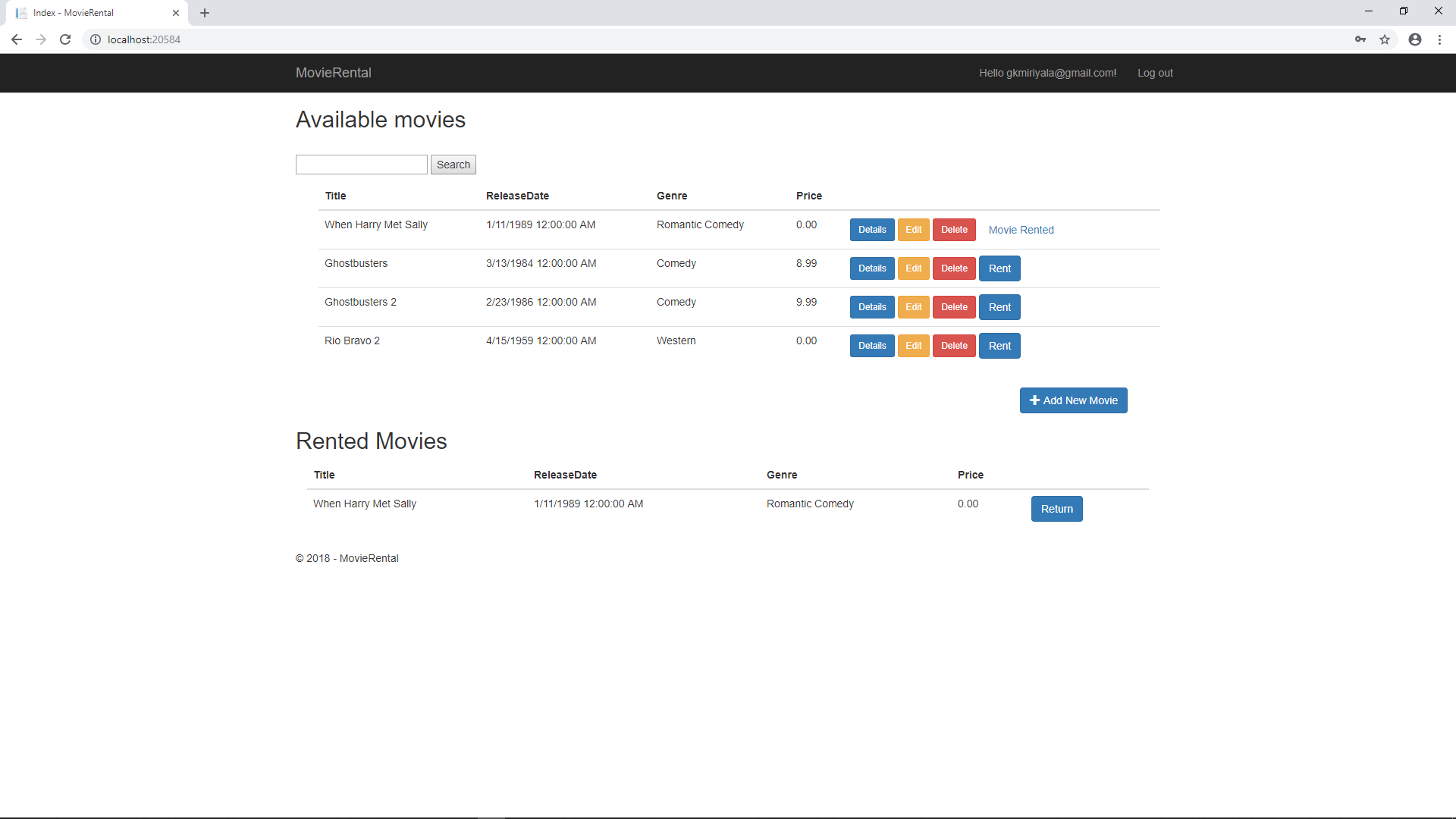
### *Access Movie List and Rent*

*Once registered and logged in, the user will be redirected to the main page where they will see a list of movies and the option to rent them if available. Each movie will be rented on a first come first serve. If a movie is rented out, the option to rent the movie will become unavailable until the movie is returned. The page displayed will also depend on the User role. “Regular Customers” will only be able to rent and return movies while “Administrators” will be able to also add, edit and delete movies.*

*Regular Customer View*

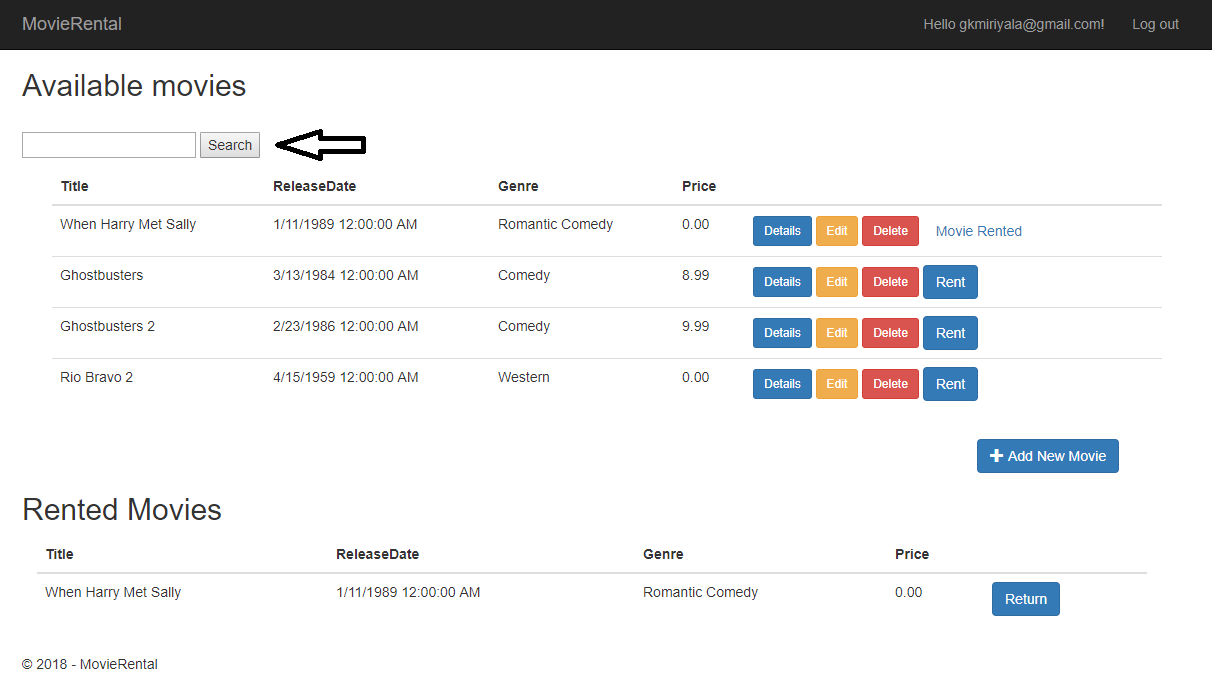


*Administrator View*



## *Reports*

1. *To access the reporting feature, from the Movie Rental Application. Click on the Search Text field on the top left after logging in*



1. *Type in the part of the movie names that you are searching for and click search to filter out movies with the specified words. Example below shows “When Harry Met Sally” after typing in “When” in the Search box*

