

San Jose State University  
College of Engineering  
Computer Engineering Department  
CMPE 273 – Enterprise Distributed Systems  
(Class Project)



Video Library Management System

**Project Report**

**Team 2**

**Abhinav Patil**  
**Venkata Karthik Mudiam**  
**Nimeet Gandhi**  
**Nilay Shah**  
**Vishal Sharma**  
**Yogesh Naik**

## **Contribution of team members.**

**Abhinav Patil:** Designing of the Web services, servlets, and unit testing.

**Neemit Gandhi:** Setting up of the Google SVN repository, Connection pooling, Working with web services and servlets.

**Yogesh Naik:** Designing of the User Interface (Admin Part), its validation, and servlets, project report, and slides.

**Nilay Shah:** Designing of User Interface (Client part) and its validation, and servlets.

**Karthik MV. :** Designing of Web services, servlets, stored procedures.

**Vishal Sharma.:** Designing of Database, JUnit configuration and testing, servlets, Project report.

## Object management Policy:

**Abstract:** We propose to discuss a solution to a very predictive problem in our project. Video Library Management System is a 3-tier web application which deals with renting movies to its customers. It deals with keeping a record of its entire customer's details, so does it for the movies. There are some rules for particular customers; the web application comfortable manages that too. It keeps the track of all the movies rented by the customers, further taking care of all the transactions of the rent. Developing the website was not a big deal; the real challenge was in making it work for mammoth clientele. Any website works fine when deployed, but a real good website will survive if it manages the increase in data and traffic, when it's scalable. The database is capable of dealing with huge data. That's when the real performance of a system is tested. We have tried our best to make the project scalable, efficient and reliable.

### Introduction

The most common term used these days is, "It is the world of internet." With the advanced technologies available, people are finally convinced to trust the internet by giving their personal details (such as their credit card information), may it be for online shopping, or anything else. Entertainment sector has benefited the most from this. This is what the report is about, an entertainment application running on this internet, but with a risk of dealing with very critical and crucial data. Such web app has to be really safe, convenient and efficient in order to run its business successfully. Developing a website is not a big deal today, it's the performance that matters the most. It's the data it manages to handle that matters. The true test is when a web application gets thousands of requests, increasing the network traffic, creating the challenges for the web app to perform. The website further allows the clients to become its members, further serving them for what they want. It maintains and stores every client's details and all the transactions done by them. It maintains the log of all the movies client rented. It also takes client's important information such as their credit card and Social Security and further keeps it safe and secure. It also keeps the log of all the movies it has, and flexibly handles it according to the business. But the question is how will it succeed? It will succeed by providing the best possible service to its clients, by giving them something more, something special than what other organizations are giving them. It is also said that "Well begun is half done". So when the organization succeeds in getting clientele, it should be able to maintain it. This is where the real challenge comes. It is hard to get the customer base of 10,000. But it is much harder to maintain it. This is where the performance characteristics come into picture. A web app should be scalable; it should be able to handle thousands of customers with any efforts. It should be able to store their information in the database. The database Schema has to be well designed and deployed for that off course. The server should be able to handle multiple requests at a time, without degrading its performance. To attract the customers, it should provide economical and attractive service to its customers by providing them with a versatile collection of movies.

### **Implementation.**

Clients will be distinguished mainly into 2 types; in other words, they'll be served on the basis to the membership they choose to have, Simple customers and Premium members. Simple customers can rent a movie and have a limitation to rent 2 movies at a time. On the other hand premium members have the freedom to have 10 movies at a time. Simple customers will have to pay the rent for the movie as soon as they take one, whereas premium one has to pay the subscription fees (monthly, quarterly, yearly, etc.) Both of them have to sign up to avail the features. They have to give all their personal and professional information (if needed) while signing up. They can choose any movie from a huge variety of collection. They can access their log anytime. They have to pay the fine if they don't return the movie on time. This information is nothing but the problem statement, and as we've implemented and developed it, it would be fair to call it as solution. Our system has an Admin to maintain it; he takes care of the users and movies if required. He can add, edit, update and delete any information about the movies as well as the users. Users can see the most favorite or high rated movies. They have a choice to get all the detailed information about the movie. Their information and logs are absolutely safe and secure. This is what makes the website good at present. Making it better and further best would be the real challenge.

### **Performance**

We've already had a discussion of being a good website. In order to make it better, it was absolutely essential for us to take care of some important technical details. We were well aware of the fact that these details would make our system much better and reliable in performance, further increasing its scalability and robustness. As mentioned in the previous sections, we needed to maintain huge amount of clients and their data. The database should work without any problems further handling bulk of data. It should also work taking less time in order to execute multiple queries. Connection pool is considered to be the best solution for this. It works in the following way.. When a client requests the database, it creates a new connection it accesses the database and after it is done with its work, the connection is closed. This happens for every request; imagine a case when there will be thousands of clients request for accessing the database. As the number of clients increase the time required is increased, so are the resources utilized. This is not considered to be an efficient working; this is where connection pool comes into picture. They are basically used to enhance the performance of the commands executing on the database. A pool opens multiple numbers of connections according to the clients, these connections once opened are given when a client request comes/arrives, client uses the connection and later returns it to the pool making it available for further clients. This way the resources are less utilized comparatively. Also the less time is spent in opening and closing the connections. The use of connection pooling makes our system take care of heavy weight resources. Following is the code for Connection Pooling.

```
package VideoLibrary;
```

```
import java.sql.*;  
import java.util.*;
```

```
public class ConnectionPool implements Runnable {
```

Video Management Library System

```
private String driver, url, username, password;
private int maxConnections;
private boolean waitIfBusy;
private Vector availableConnections, busyConnections;
private boolean connectionPending = false;
int initialConnections = 50;
static private ConnectionPool pool_instance;

private ConnectionPool(){
    this.driver = "com.mysql.jdbc.Driver";
    this.url = "jdbc:mysql://localhost/videolibrary";
    this.username = "root";
    this.password = "root";
    this.maxConnections = 100;
    this.waitIfBusy = true;

    if (initialConnections > maxConnections) {
        initialConnections = maxConnections;
    }
    availableConnections = new Vector(initialConnections);
    busyConnections = new Vector();
    for(int i=0; i<initialConnections; i++) {
        try {
            availableConnections.addElement(makeNewConnection());
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

static public ConnectionPool getInstance()
{
    if (pool_instance == null){
        pool_instance = new ConnectionPool();
    }
    return pool_instance;
}

public synchronized Connection getConnection()
    throws SQLException {
    if (!availableConnections.isEmpty()) {
        Connection existingConnection =
            (Connection)availableConnections.lastElement();
        int lastIndex = availableConnections.size() - 1;
        availableConnections.removeElementAt(lastIndex);

        if (existingConnection.isClosed()) {
```

```
        notifyAll(); // Freed up a spot for anybody waiting
        return(getConnection());
    } else {
        busyConnections.addElement(existingConnection);
        return(existingConnection);
    }
} else {

    if ((totalConnections() < maxConnections) &&
        !connectionPending) {
        makeBackgroundConnection();
    } else if (!waitIfBusy) {
        throw new SQLException("Connection limit reached");
    }
    // Wait for either a new connection to be established
    // (if you called makeBackgroundConnection) or for
    // an existing connection to be freed up.
    try {
        wait();
    } catch (InterruptedException ie) {}
    // Someone freed up a connection, so try again.
    return(getConnection());
}
}

private void makeBackgroundConnection() {
    connectionPending = true;
    try {
        Thread connectThread = new Thread(this);
        connectThread.start();
    } catch (OutOfMemoryError oome) {
        // Give up on new connection
    }
}

public void run() {
    try {
        Connection connection = makeNewConnection();
        synchronized(this) {
            availableConnections.addElement(connection);
            connectionPending = false;
            notifyAll();
        }
    } catch (Exception e) { // SQLException or OutOfMemory
        // Give up on new connection and wait for existing one
        // to free up.
    }
}
```

```
private Connection makeNewConnection()
    throws SQLException {
    try {
        // Load database driver if not already loaded
        Class.forName(driver);
        // Establish network connection to database
        Connection connection =
            DriverManager.getConnection(url, username, password);
        return(connection);
    } catch(ClassNotFoundException cnfe) {
        // Simplify try/catch blocks of people using this by
        // throwing only one exception type.
        throw new SQLException("Can't find class for driver: " +
                                driver);
    }
}

public synchronized void free(Connection connection) {
    busyConnections.removeElement(connection);
    availableConnections.addElement(connection);
    // Wake up threads that are waiting for a connection
    notifyAll();
}

public synchronized int totalConnections() {
    return(availableConnections.size() +
           busyConnections.size());
}

public synchronized void closeAllConnections() {
    closeConnections(availableConnections);
    availableConnections = new Vector();
    closeConnections(busyConnections);
    busyConnections = new Vector();
}

private void closeConnections(Vector connections) {
    try {
        for(int i=0; i<connections.size(); i++) {
            Connection connection =
                (Connection)connections.elementAt(i);
            if (!connection.isClosed()) {
                connection.close();
            }
        }
    } catch(SQLException sqle) {
```

```
// Ignore errors; garbage collect anyhow
}
}

public synchronized String toString() {
    String info =
        "ConnectionPool(" + url + "," + username + ")" +
        ", available=" + availableConnections.size() +
        ", busy=" + busyConnections.size() +
        ", max=" + maxConnections;
    return(info);
}
}
```

Connection pooling have been proved to be very important in the industry. Following graph shows its performance characteristics.

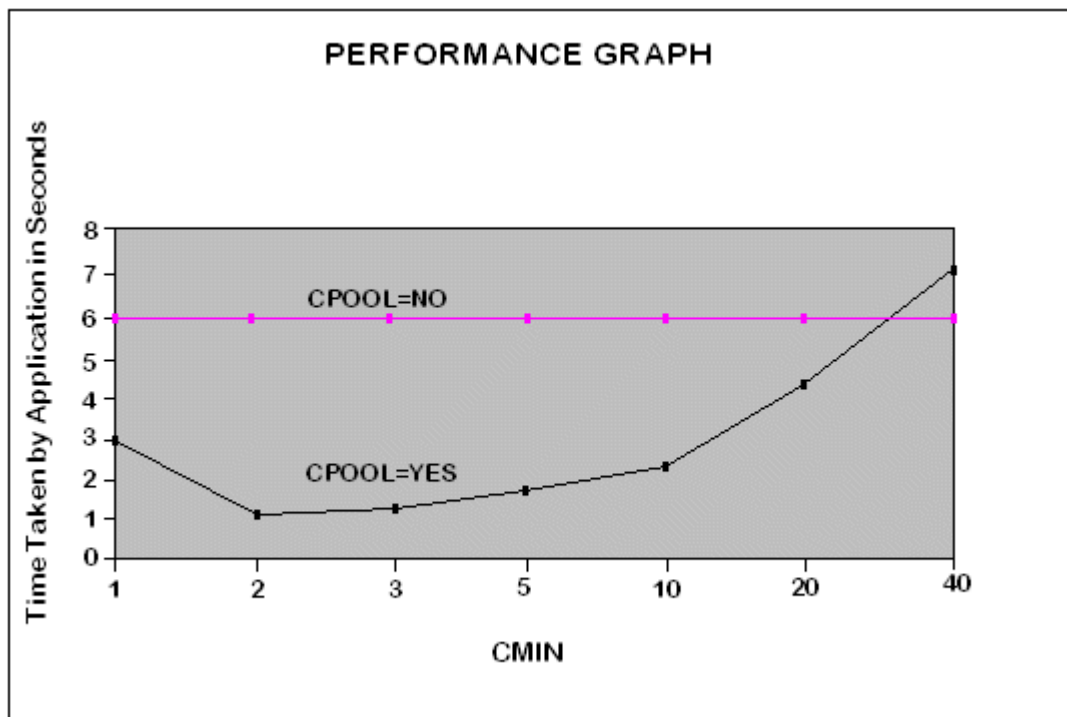


Image Source: [http://cs.felk.cvut.cz/10gr2/appdev.102/b14407/pc\\_11thr.htm](http://cs.felk.cvut.cz/10gr2/appdev.102/b14407/pc_11thr.htm)

Following are some of the Database considerations.



**Use of InnoDB database Engine over MyISAM :**

We have made sure that most of our tables that are prone to get higher hits are using InnoDB as the database engine rather than MyISAM. The reason behind this is that InnoDB uses row level locking at insertion and updating occurrences in contrast with table level locking of MyISAM engine. Using InnoDB engine will really prove to be highly efficient for our video library system where huge number of hits to the database is expected in quick succession. Using MyISAM engine would have resulted in the latency issues at the database level.

**Efficient Use of Indexing on the required fields:**

Indexing has always been one of the major considerations in the bigger projects with heavy data requirements. Not using indexing or even sometimes bad usage of indexing can result in unwanted latency issues. It can even take MySQL to a hang state. Keeping the amount of data that will be handled in the video library system we have used indexing the best possible manner, which will result in the faster retrieval of the required data thereby evading latency issues.

**Database Table details:**

**Movie:** This table holds the complete dump of movies participating in the system and all the information related to it. The “isActive” field signifies whether the particular movie is active or not. If a movie is active, this field is set to ‘T’ and when we wish to delete a movie from the system, instead of actually deleting the movie, we simply set its “isActive” field as ‘F’. It helps in a way, if we wish to add the same movie again to the system, we need not assign a new “movieId” to it. Our job is done only by setting “isActive” field to ‘T’ again.

```
mysql> desc movie;
```

Field	Type	Null	Key	Default	Extra
movieId	bigint(20)	NO	PRI	NULL	auto_increment
movieName	varchar(45)	YES	MUL	NULL	
movieDesc	varchar(500)	YES		NULL	
releaseDate	varchar(25)	YES		NULL	
movieCategoryId	int(11)	YES	MUL	NULL	
movieBanner	varchar(45)	YES	MUL	NULL	
availableCopies	int(11)	YES		NULL	
movieRating	int(5)	YES		NULL	
movieRent	int(11)	YES		NULL	
movieHits	int(10)	YES		NULL	
isActive	varchar(45)	YES	MUL	NULL	

**Movie Category:** This table contains the all the categories of the movies that are available in the video library system. The categories are like – action movies, horror movies, romantic movies and so on.

```
mysql> desc movieCategory;
```

Field	Type	Null	Key	Default	Extra
movieCategoryId	int(11)	NO	PRI	NULL	auto_increment
movieCategoryName	varchar(45)	YES		NULL	

**User details:** This table contains all the user specific information. The “isActive” field signifies whether the user is active or not. It will have value ‘T’ for active and ‘F’ for inactive users or the users that have unsubscribed. If an unsubscribed user wants to subscribe again, in that case we simply set “isActive” field to ‘T’ instead of assigning new membershipId to him.

```
mysql> desc userDetails;
```

Field	Type	Null	Key	Default	Extra
membershipId	int(11)	NO	PRI	NULL	
emailId	varchar(45)	YES		NULL	
password	varchar(45)	YES		NULL	
firstName	varchar(45)	YES		NULL	
lastName	varchar(45)	YES		NULL	
dateOfBirth	varchar(45)	YES		NULL	
address	varchar(100)	YES		NULL	
city	varchar(45)	YES		NULL	
state	varchar(45)	YES		NULL	
zipCode	int(11)	YES	MUL	NULL	
gender	varchar(1)	YES		NULL	
userType	varchar(1)	YES		NULL	
expiryDate	varchar(45)	YES		NULL	
registrationDate	varchar(45)	YES		NULL	
isActive	varchar(1)	YES	MUL	NULL	

**Premium customer:** This table holds the complete dump of users those have opted for the premium membership of the video library system.

```
mysql> desc premiumcustomer;
```

Field	Type	Null	Key	Default	Extra
premiumCustomerId	int(11)	NO	PRI	NULL	auto_increment
membershipId	int(11)	YES	MUL	NULL	
subscriptionId	int(11)	YES	MUL	NULL	
plan	varchar(25)	YES	MUL	NULL	
numberOfMoviesIssued	int(11)	YES		NULL	
paymentStatus	varchar(1)	YES	MUL	NULL	

**Simple customer:** Table simplecustomer contains the information about all the users those have not opted for the premium membership of the video library system.

```
mysql> desc simplecustomer;
```

Field	Type	Null	Key	Default	Extra
simpleCustomerId	int(11)	NO	PRI	NULL	auto_increment
membershipId	int(11)	YES	MUL	NULL	
numberOfMoviesIssued	int(11)	YES		NULL	

**Rental transactions:** Table rentaltransaction will hold all the transaction related information that are needed when a movie is rented from the video library system. Field “membershipId” contains the membershipId of the user who has rented the movie. When the movie is rented the field “isActive” is set to ‘T’ which signifies that the particular movie is yet to be returned. And once the movie is returned the “isActive” field is set to ‘F’ which means that the particular movie has been returned and the transaction is closed.

```
mysql> desc rentaltransactions;
```

Field	Type	Null	Key	Default	Extra
rentaltransactionId	int(11)	NO	PRI	NULL	auto_increment
membershipId	int(11)	YES	MUL	NULL	
movieId	int(11)	YES	MUL	NULL	
issueDate	varchar(25)	YES		NULL	
dueDate	varchar(25)	YES		NULL	
actualReturnDate	varchar(25)	YES		NULL	
fineAmount	int(11)	YES		NULL	
isActive	varchar(1)	YES	MUL	NULL	
transactionId	bigint(12)	YES	MUL	NULL	

**Subscription plan:** Table subscriptionplan holds all the plans available for subscription for the users and the corresponding charges for each plan. A user can subscribe any of the plans listed in this table but has to pay the charged amount associated with the specific plan .

```
mysql> desc subscriptionplans;
```

Field	Type	Null	Key	Default	Extra
subscriptionId	int(11)	NO	PRI	NULL	auto_increment
plan	varchar(25)	YES		NULL	
charges	int(11)	YES		NULL	

**User accounts history:** Major purpose of this table is to show the users accounts details and its history.

```
mysql> desc useraccounthistory;
```

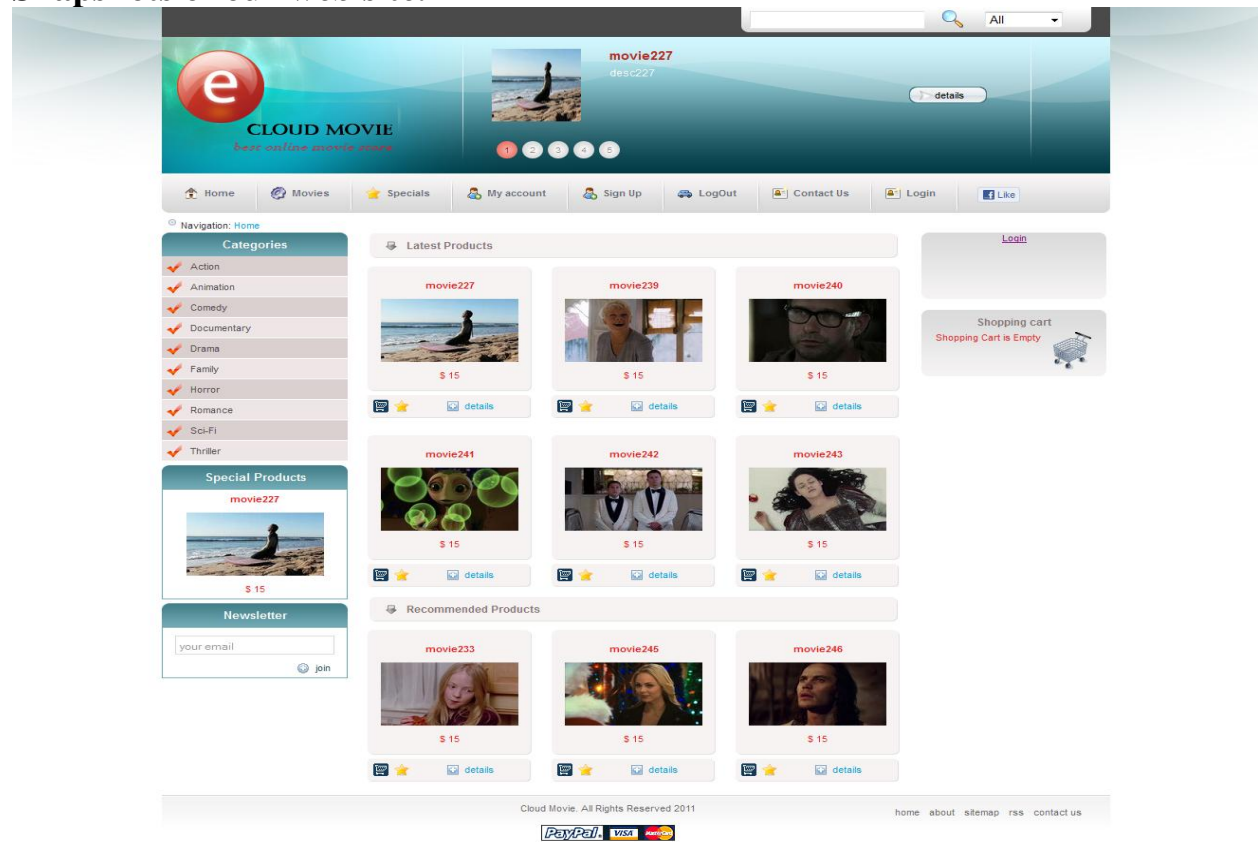
Field	Type	Null	Key	Default	Extra
userAccountHistoryId	int(11)	NO	PRI	NULL	auto_increment
transactionId	int(11)	YES	MUL	NULL	
userType	varchar(1)	YES		NULL	
plan	varchar(40)	YES		NULL	

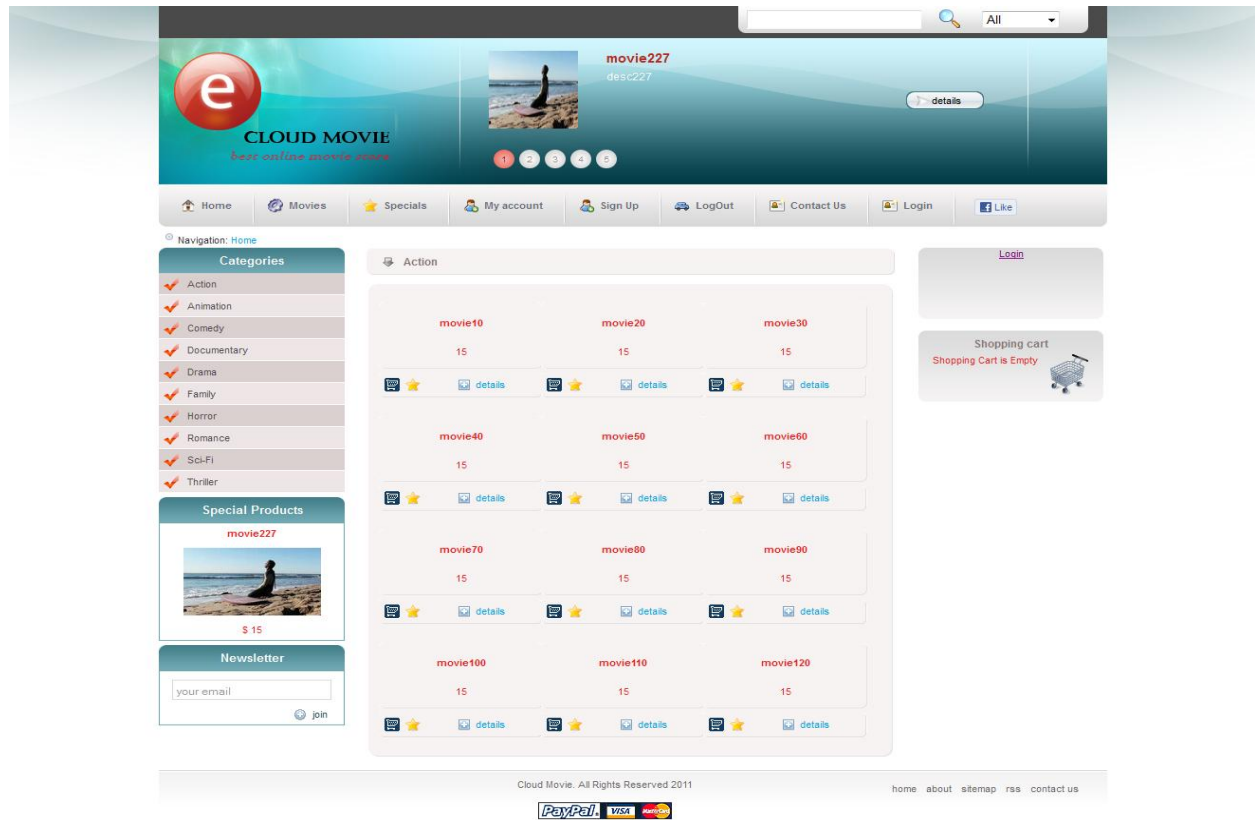
```
mysql> desc transactions;
```

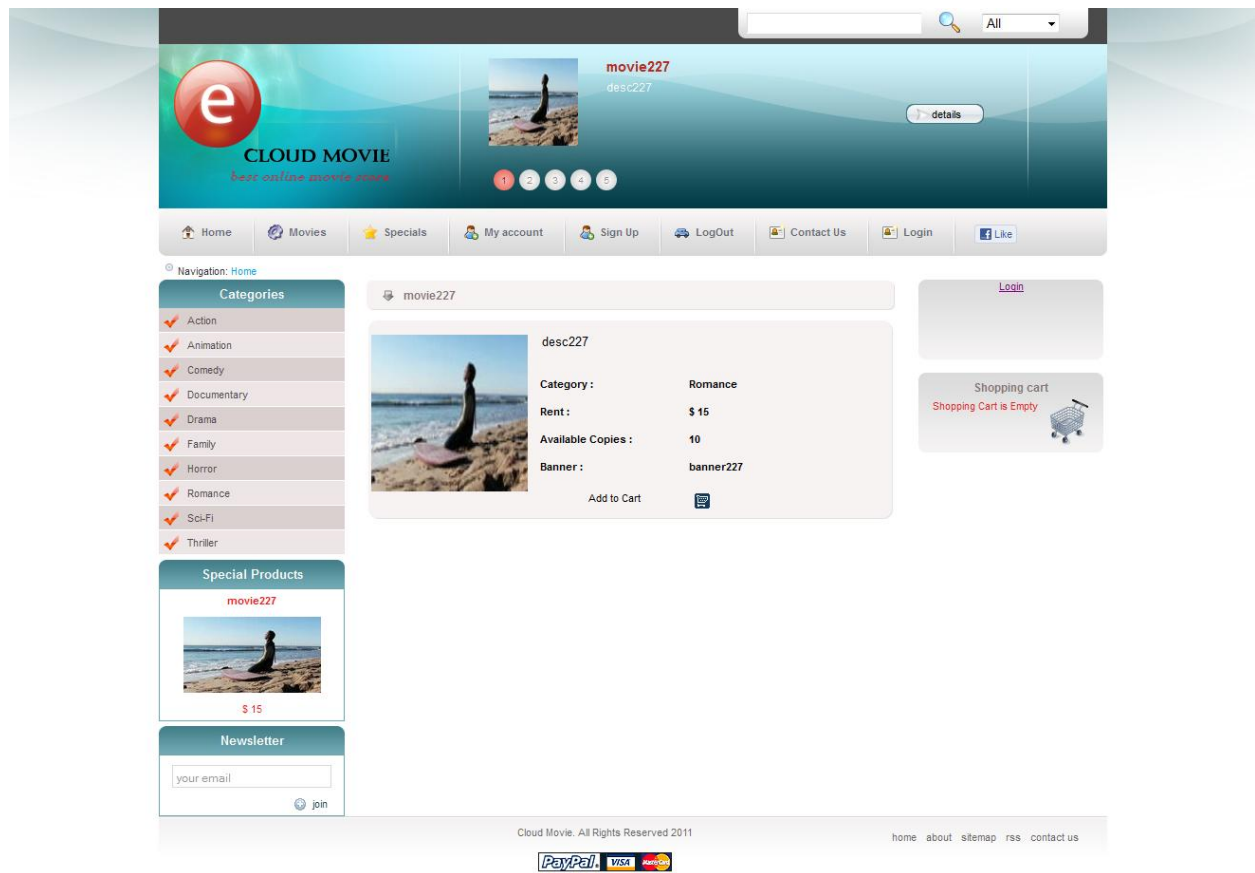
Field	Type	Null	Key	Default	Extra
transactionId	int(11)	NO	PRI	NULL	auto_increment
membershipId	int(11)	YES		NULL	
amountPaid	int(11)	YES		NULL	
transactionDate	varchar(45)	YES		NULL	

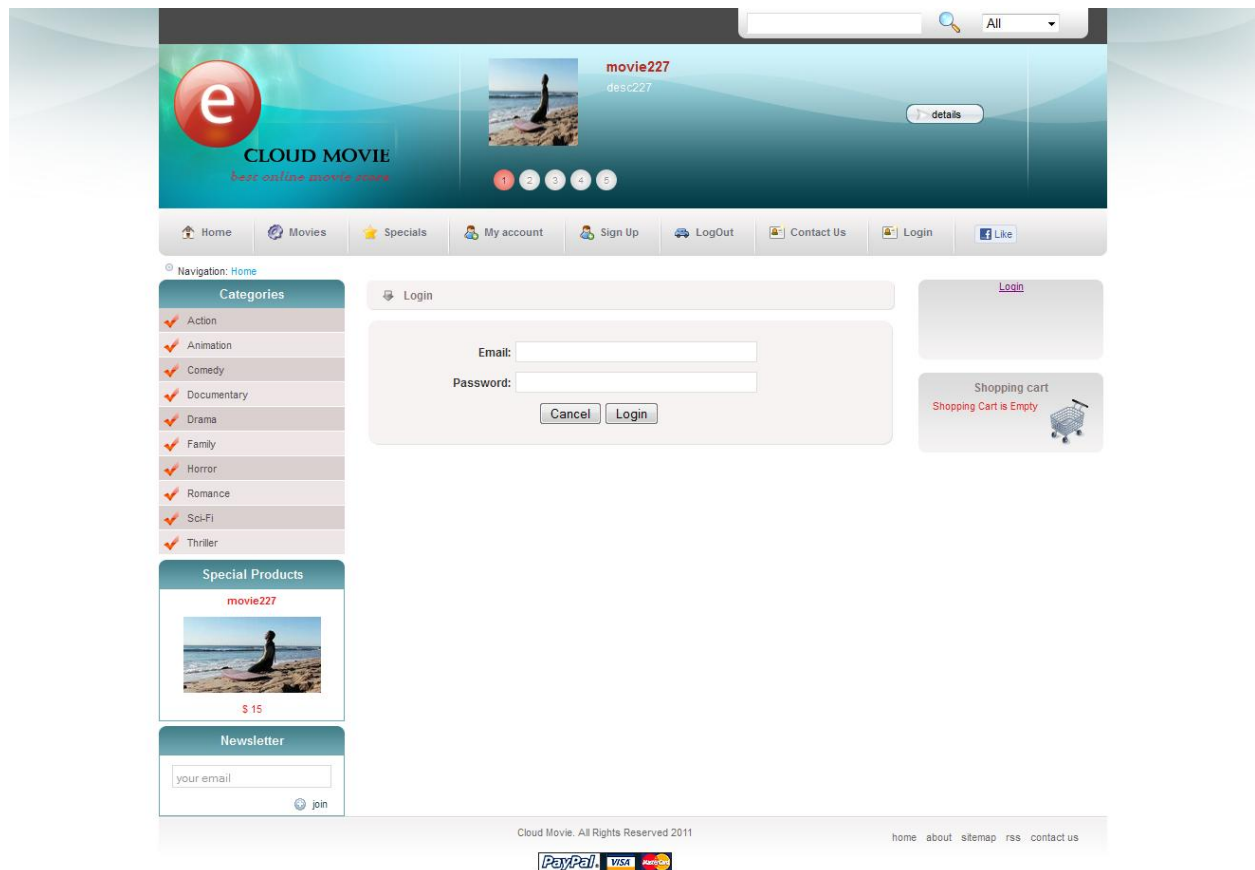
Web services also have a specialty of dealing with the clients as the number of requests grows. This is how it works, as the number of clients request increase, the load obviously increases, web services deals with this situation by creating threads in order to handle those requests. In simple words, more the client request, greater amount of threads are created. This makes web service efficient in the middle tier.

## Snapshots of our web site.

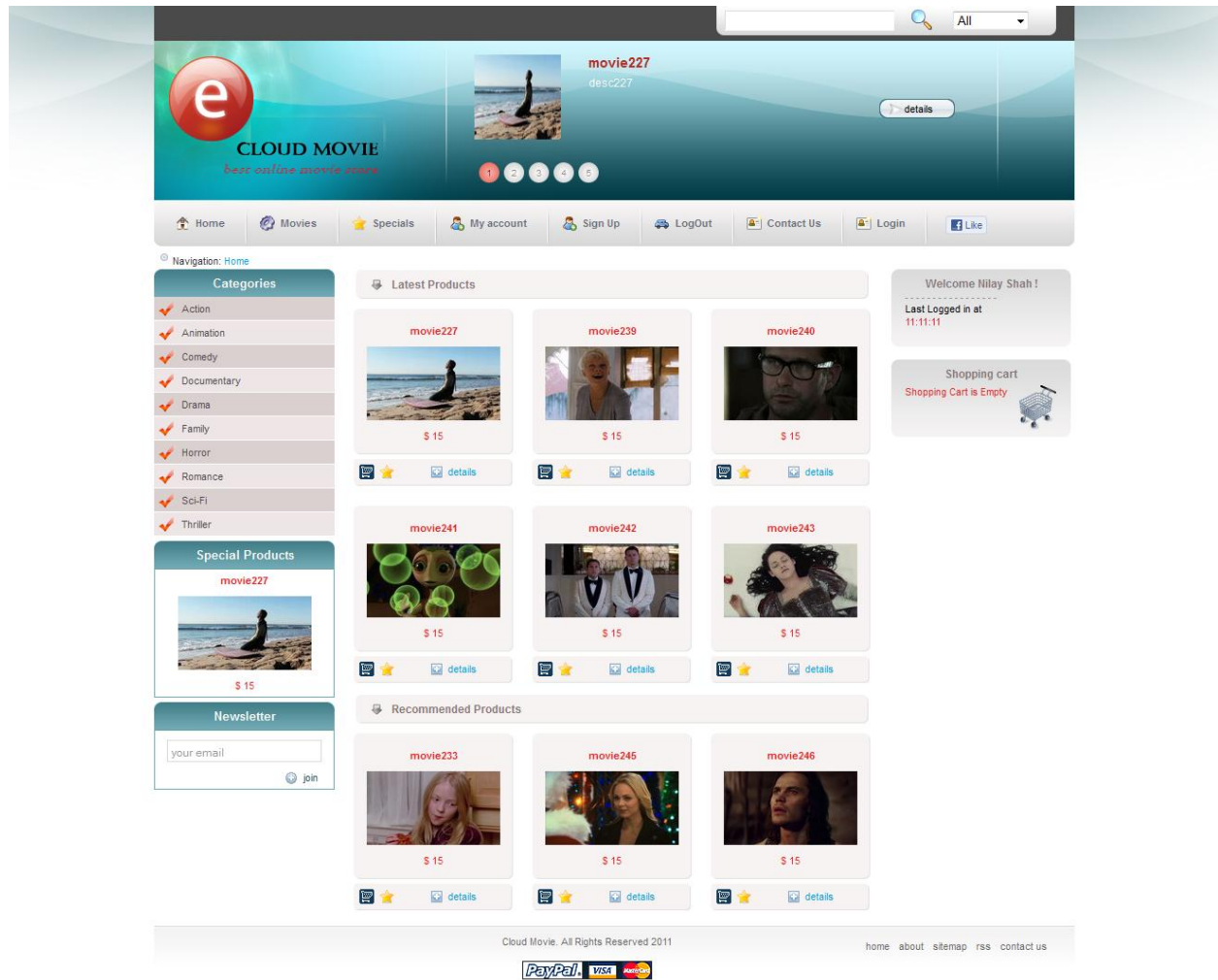


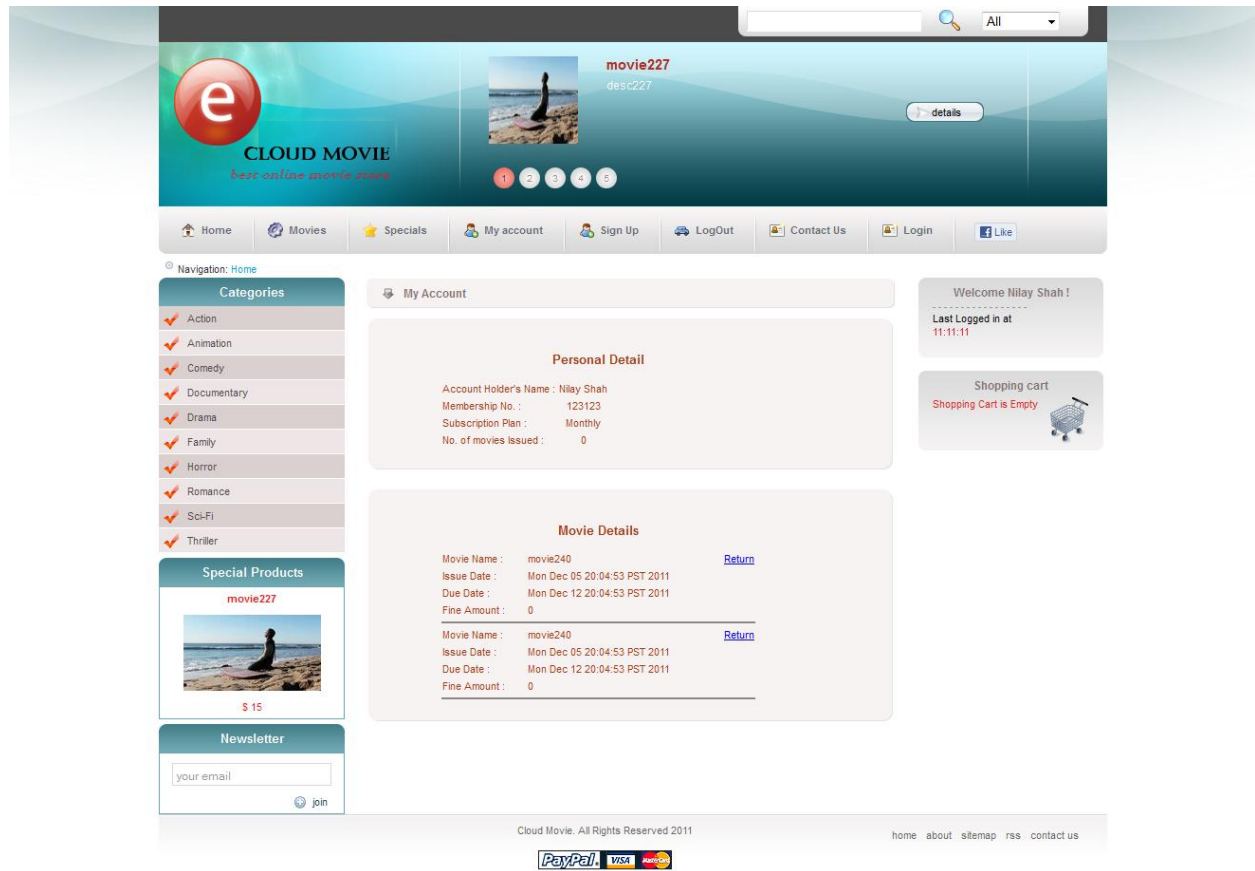


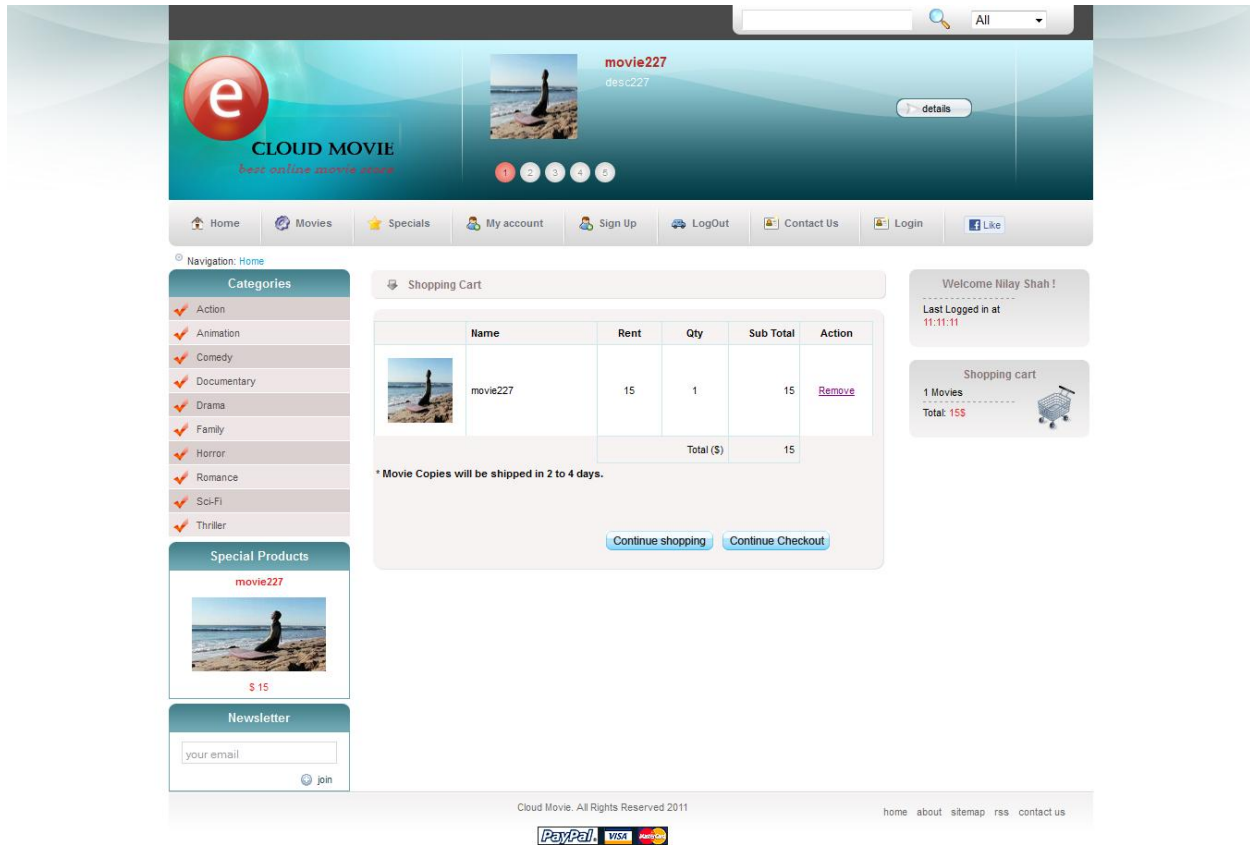


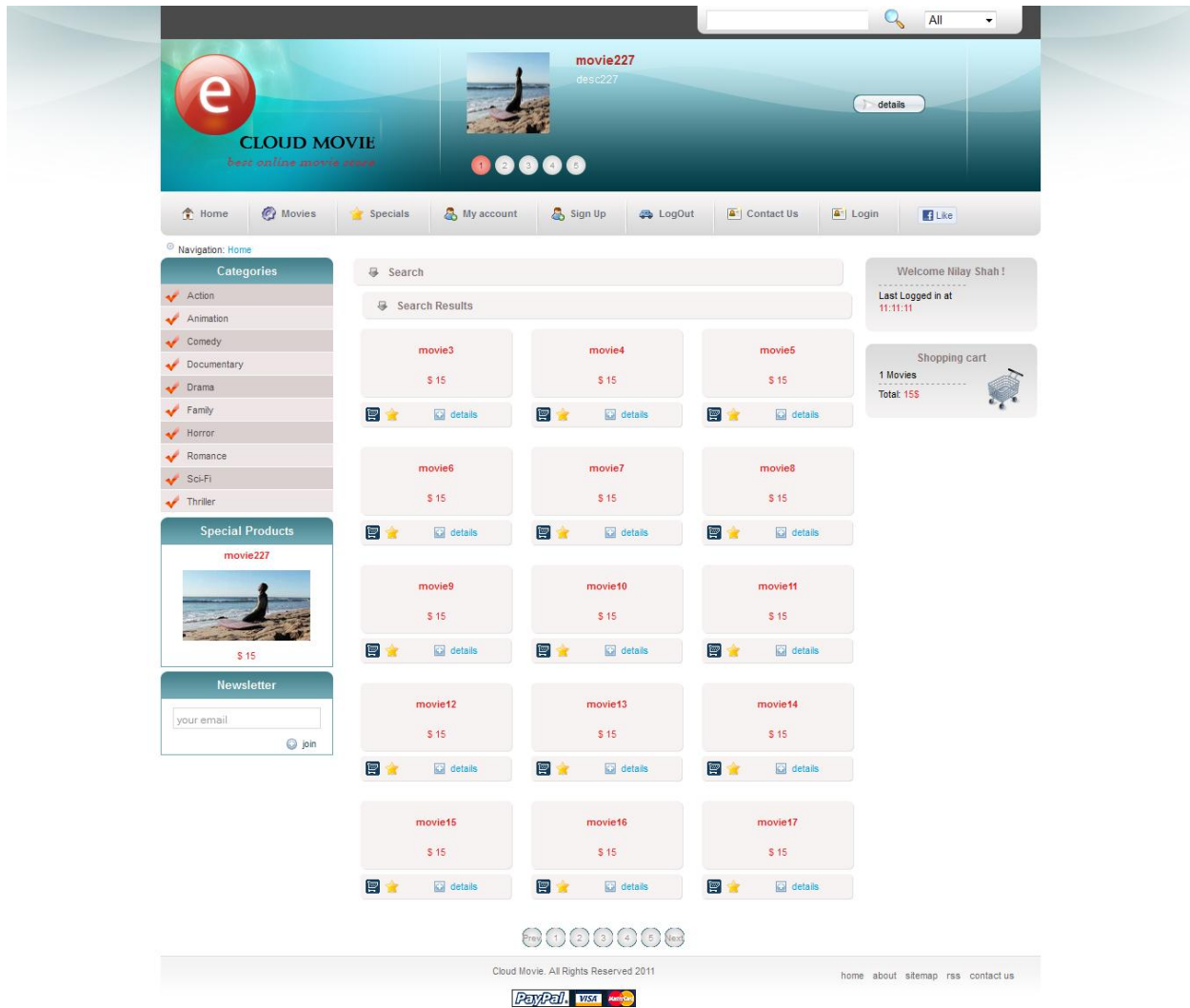


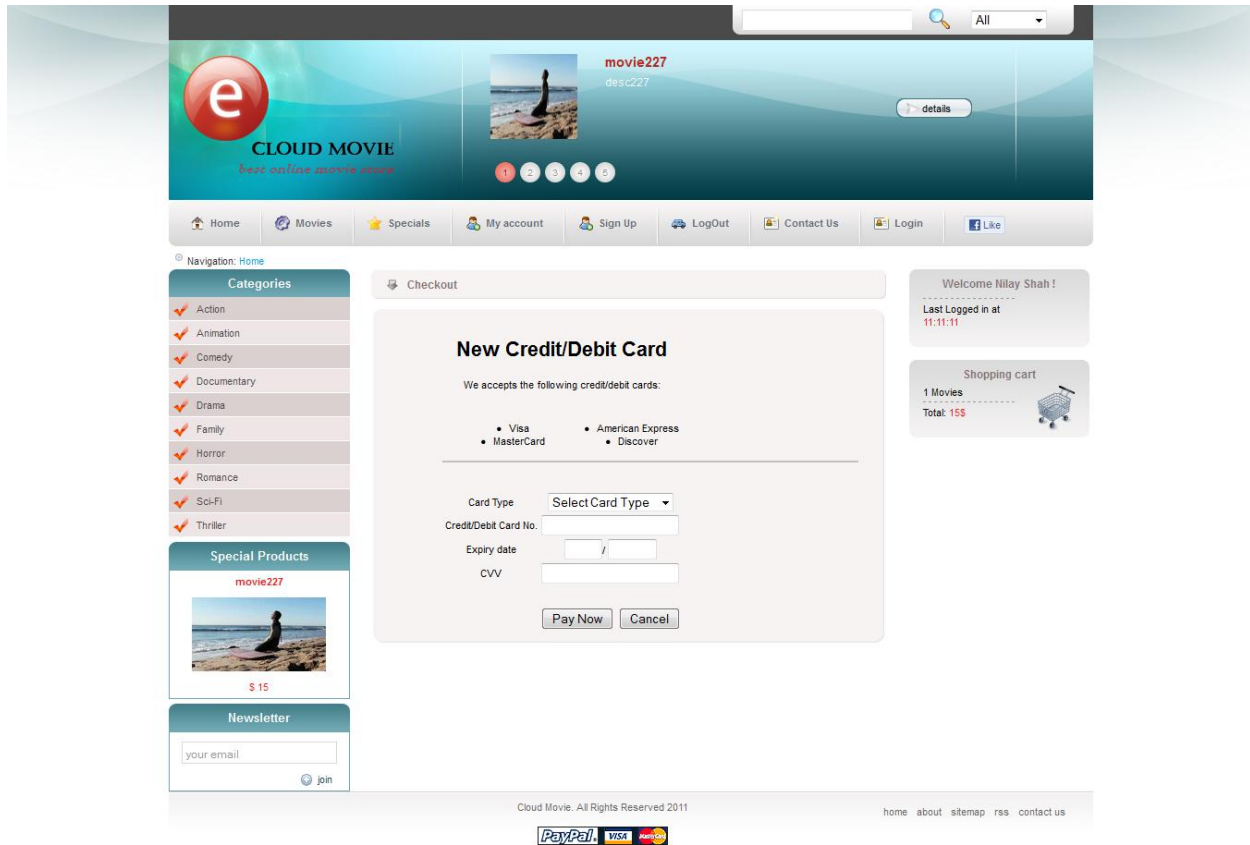


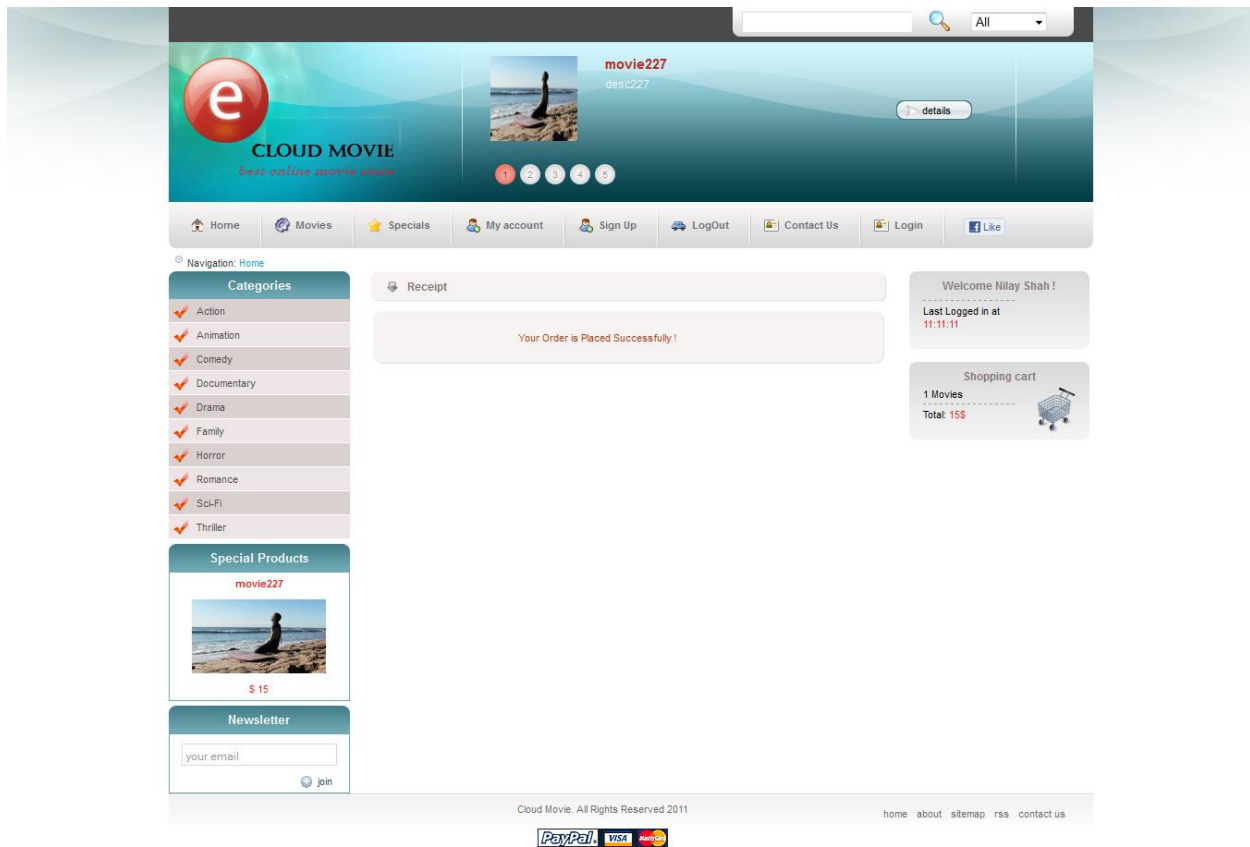












Home

Movies List

Regular Customers List

Premium Customers List

ADD MOVIE

Sr.	Movie Name	Available Copies	Rent	Edit	Delete
1	movie3	10	15	Edit	Delete
2	movie4	10	15	Edit	Delete
3	movie5	10	15	Edit	Delete
4	movie6	10	15	Edit	Delete
5	movie7	10	15	Edit	Delete
6	movie8	10	15	Edit	Delete
7	movie9	10	15	Edit	Delete
8	movie10	10	15	Edit	Delete
9	movie11	10	15	Edit	Delete
10	movie12	10	15	Edit	Delete
11	movie13	10	15	Edit	Delete
12	movie14	10	15	Edit	Delete
13	movie15	10	15	Edit	Delete
14	movie16	10	15	Edit	Delete
15	movie17	10	15	Edit	Delete

Home

Movies List

Regular Customers List

Premium Customers List

Add Movie

Movie Name:

Movie Description:

Release Date:

Movie Category:

Movie Banner:

Available Copies:

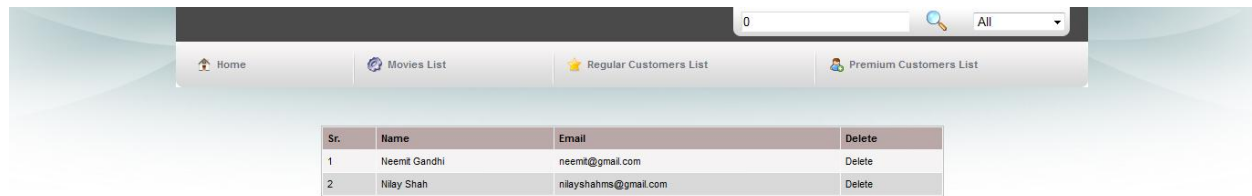
Rent Amount:

Active : ☒ Active ☐ Inactive

Submit

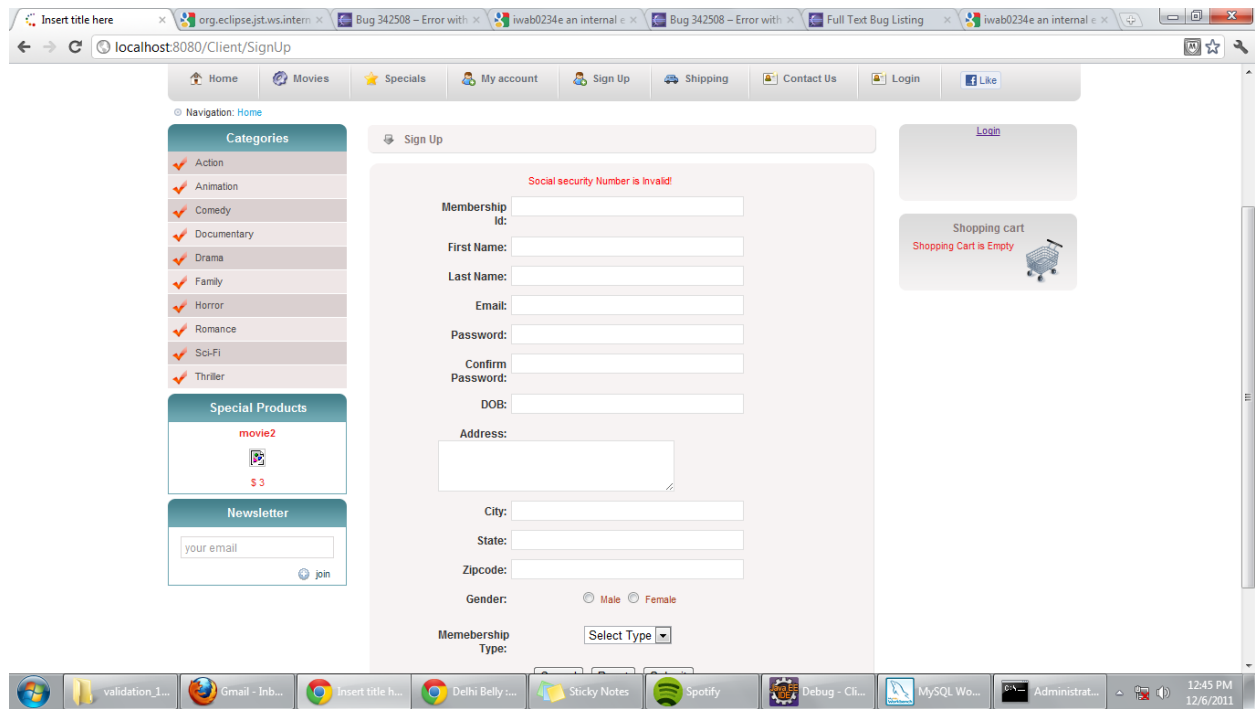
Reset

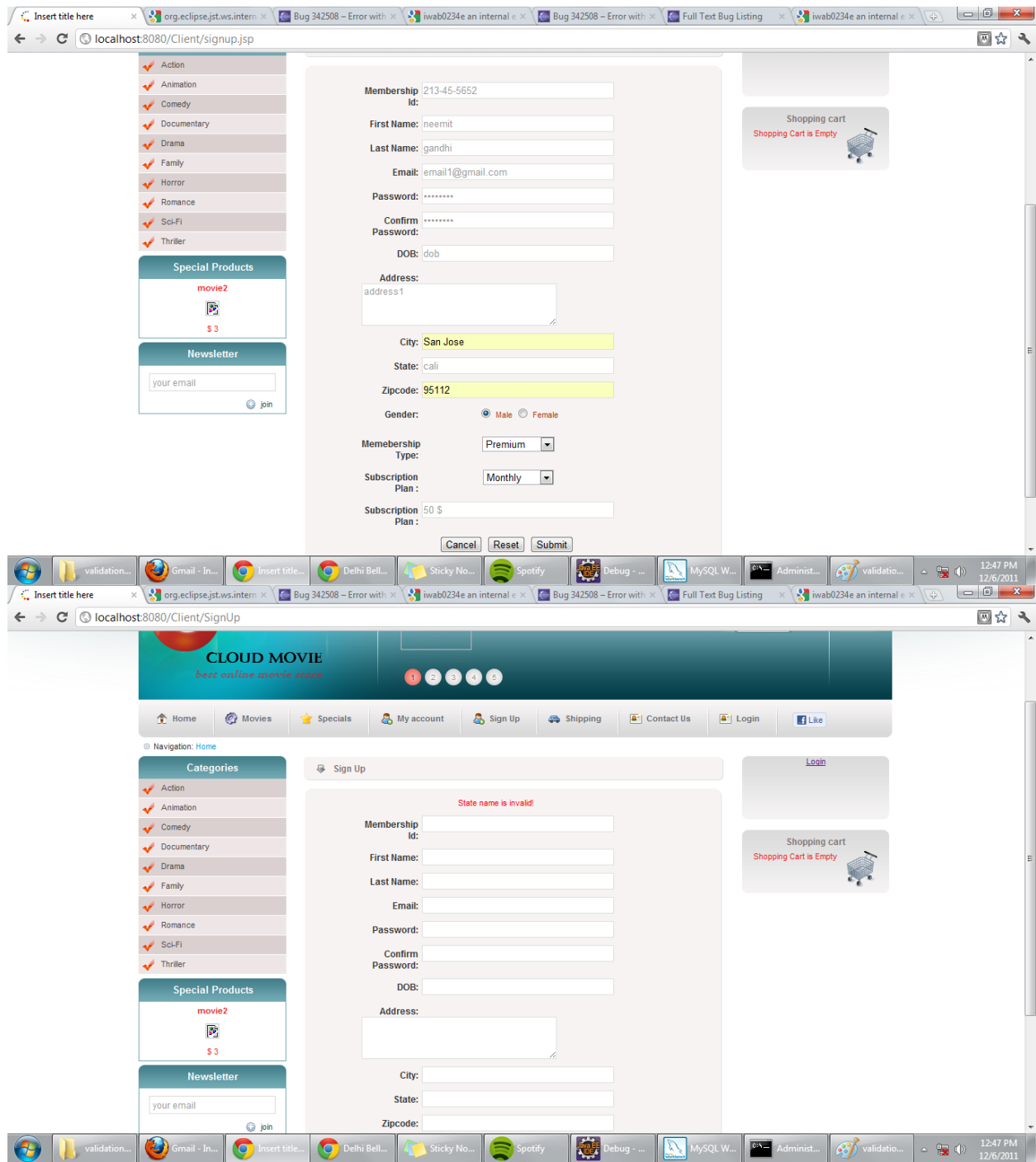
Cancel

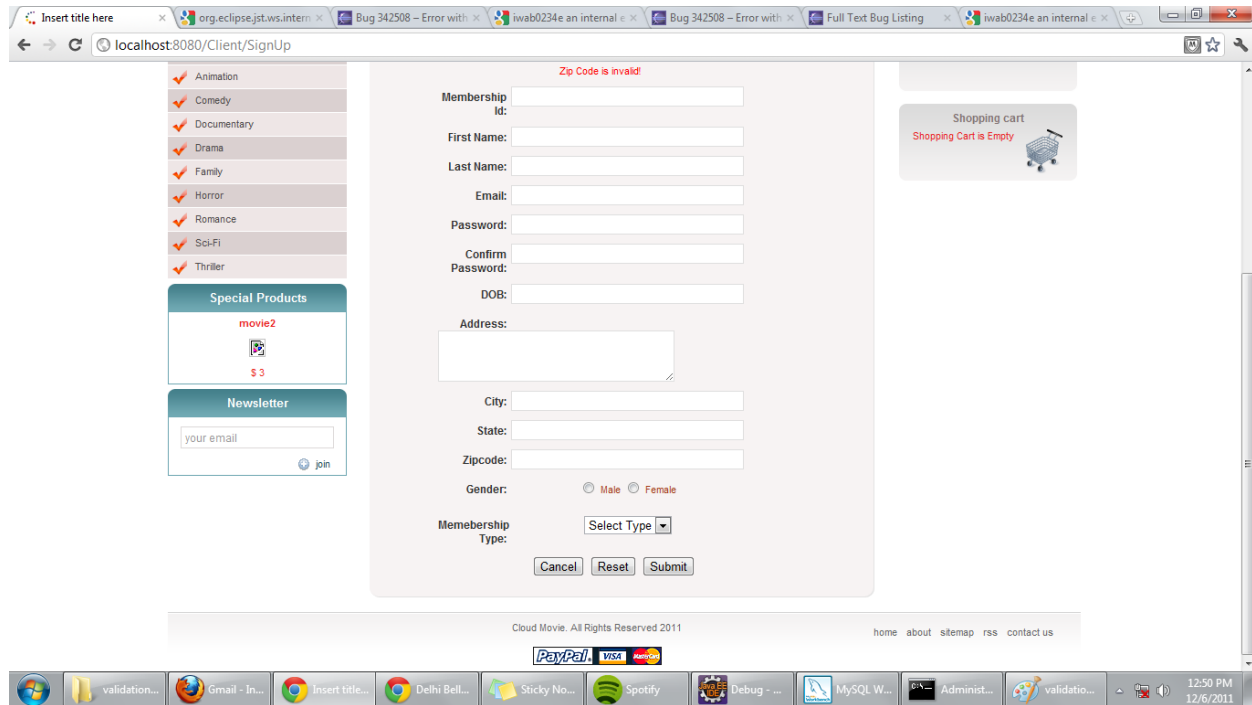


Sr.	Name	Email	Delete
1	Neeml Gandhi	neeml@gmail.com	Delete
2	Nilay Shah	nilayshahms@gmail.com	Delete



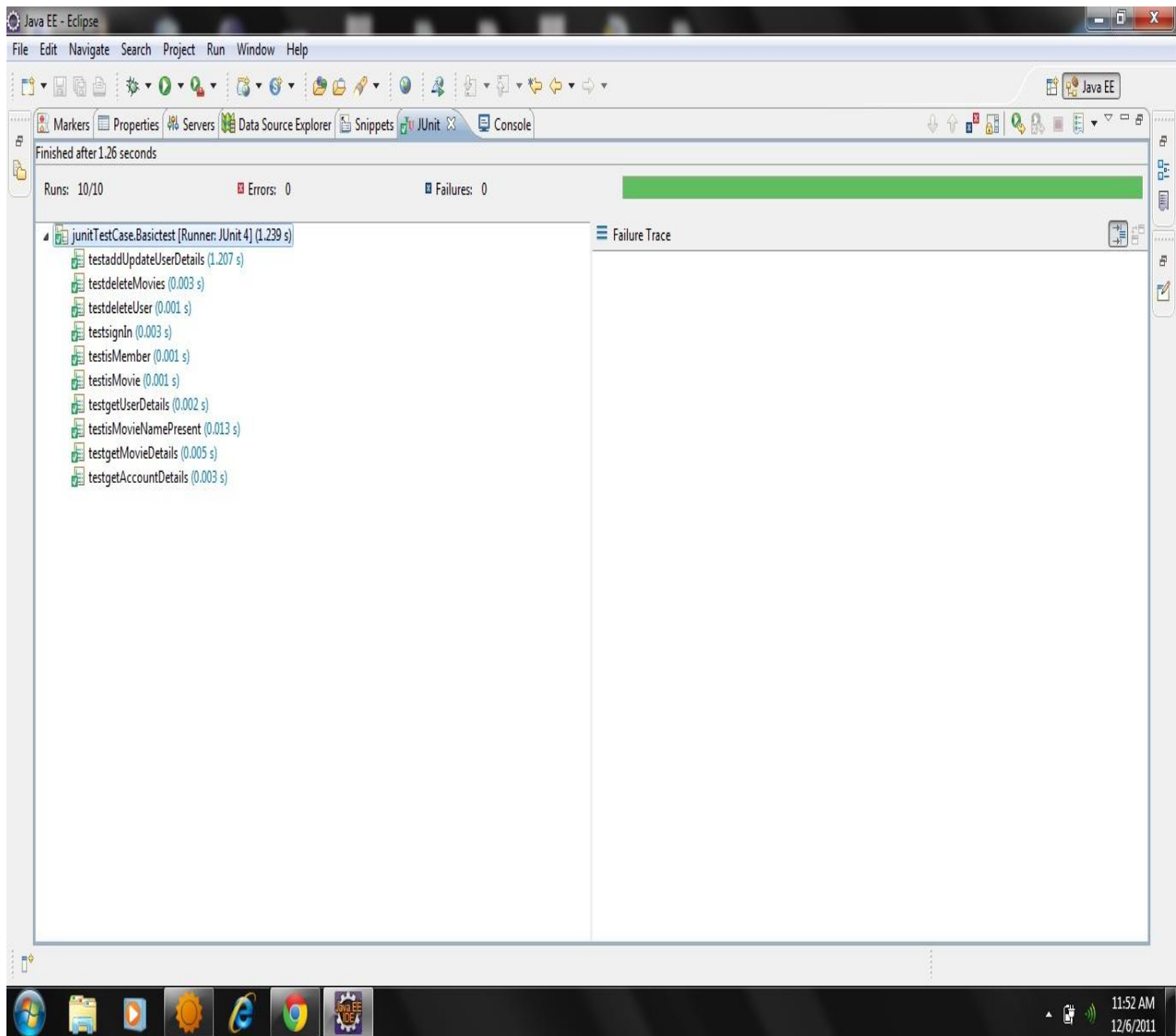






## Testing

Testing is an integral part of any software development cycle. It assures the quality and functionality of the software. According to the sources on the internet, at least 70% of the time is to be spent on testing. One can imagine the importance of it. A product has to be thoroughly tested in order to be recognized in the market and further establishing itself. As a software development team, we've checked the functionality and performance of our system, we have focused more on unit testing. We have also studied the performance characteristics. It's a way to check and test individual units of source code which further helps in determining whether those methods are fit for use. Following is the output of the test cases.



## Conclusion

We have implemented a video library management system which deals with the creating members and movies, dealing with them and all their information in very minute details. Does and maintain all the transactions related to renting the movies. We also managed to make our system scalable and reliable by increasing its performance using connection pooling, and InnoDB. We have further managed to deal with heavy resources using the same. Our system is capable of maintaining huge database, we currently have 10,000 premium and simple customers, and one million movies in our database.

### **Observation and lessons learned.**

Firstly, web services play a crucial part in the middle tier architecture. Servlets and jsp pages help it work efficiently, Connection pooling helps the database function properly, and more importantly, it helps reducing resource load. In other words it helps handling heavy weight resources. InnoDB was pretty useful in terms of accessing the database frequently, in other words, it was useful in the situations where the system frequently updates and inserts values in the database. InnoDB was also useful as data integrity was our first priority. Google SVN repository also helps in keeping the code up to date and always available for the team. Which we believe is absolutely convenient. The very important observation the comparison of the performance with and without connection pool. Besides this, we also enhanced our team learning and software development skills, further dealing with time management.