1. Introduction

Movie Central is a subscription-based media service application which offers online streaming of movies. This service has two types of users, admins and customers. Admin has rights to manage movies and to view customer or movie stats. A registered customer can watch movies and subscribe himself for watching paid movies. This application is a Restful web application with Spring Boot framework. The User Interface is developed with React JS and Redux while, database is hosted on Amazon RDS SQL server. The application is hosted on URL : http://ec2-34-209-69-136.us-west-2.compute.amazonaws.com

2. Bonus Feature Implemented

1. Movie Recommendation: Movie recommendations are showed under the tab 'Recommended for you'. The recommendations are based on the movies customer has watched. On the basis of particular director's or actor's movie customer has watched, other movies directed by the same director or actor are showed as recommendations to customer.
2. Implemented elastic search for full text search.

3. High Level and Component Level Design

High level design for the application is:

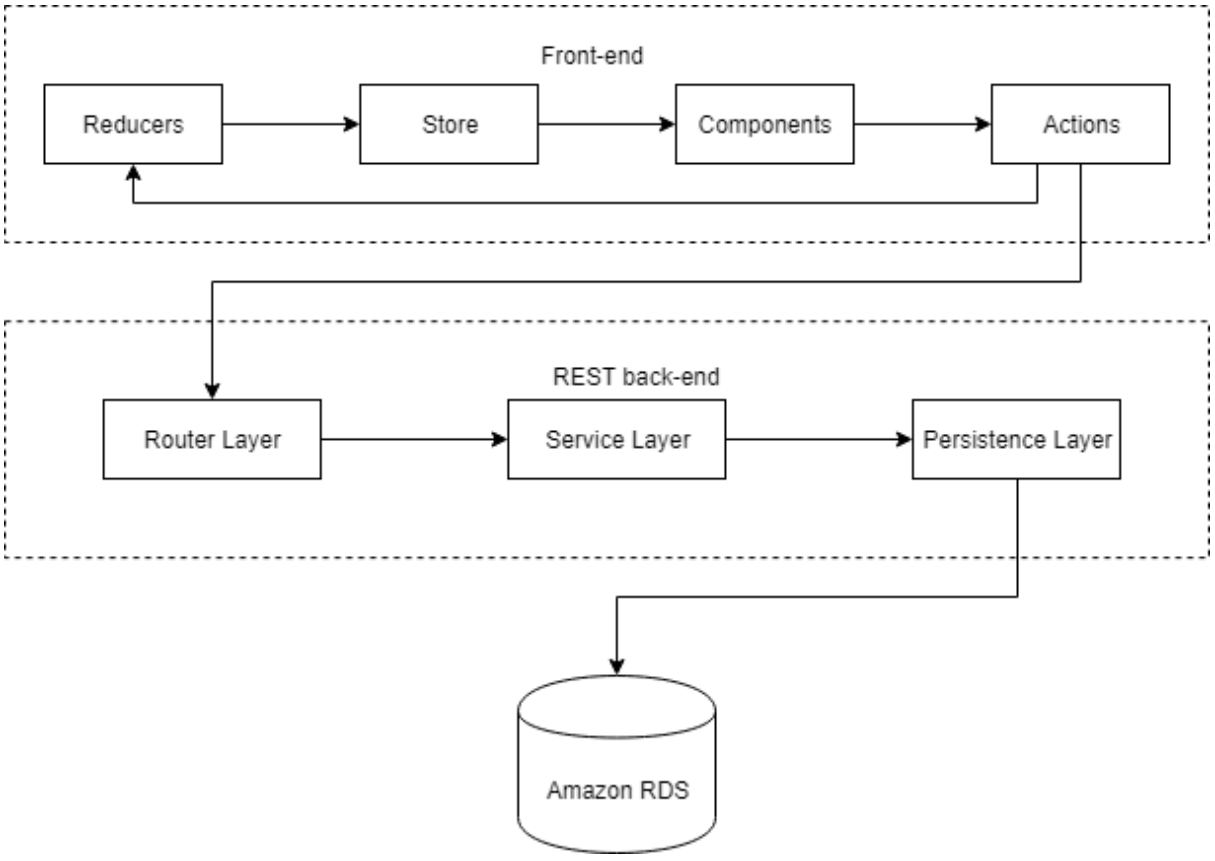


Fig 3.1: High Level Design

Component level design is:

Fig 3.2: Component Level Design

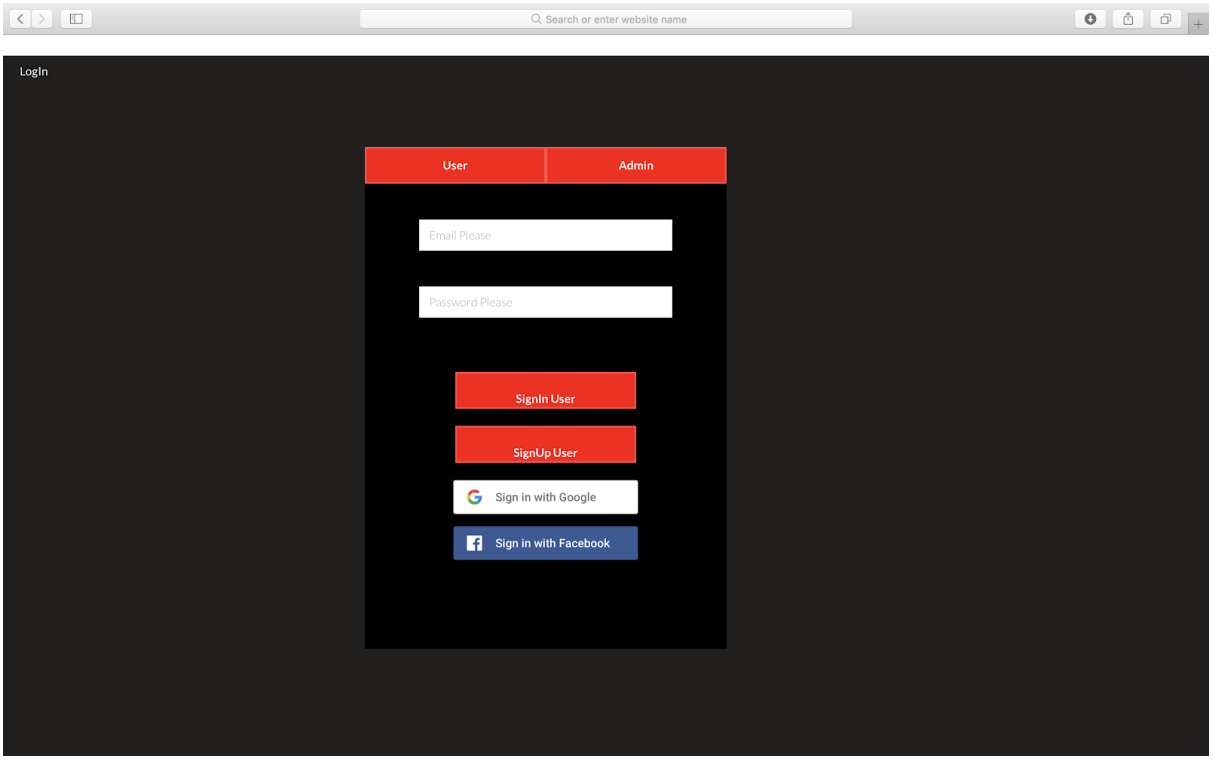
4. Technology Choices

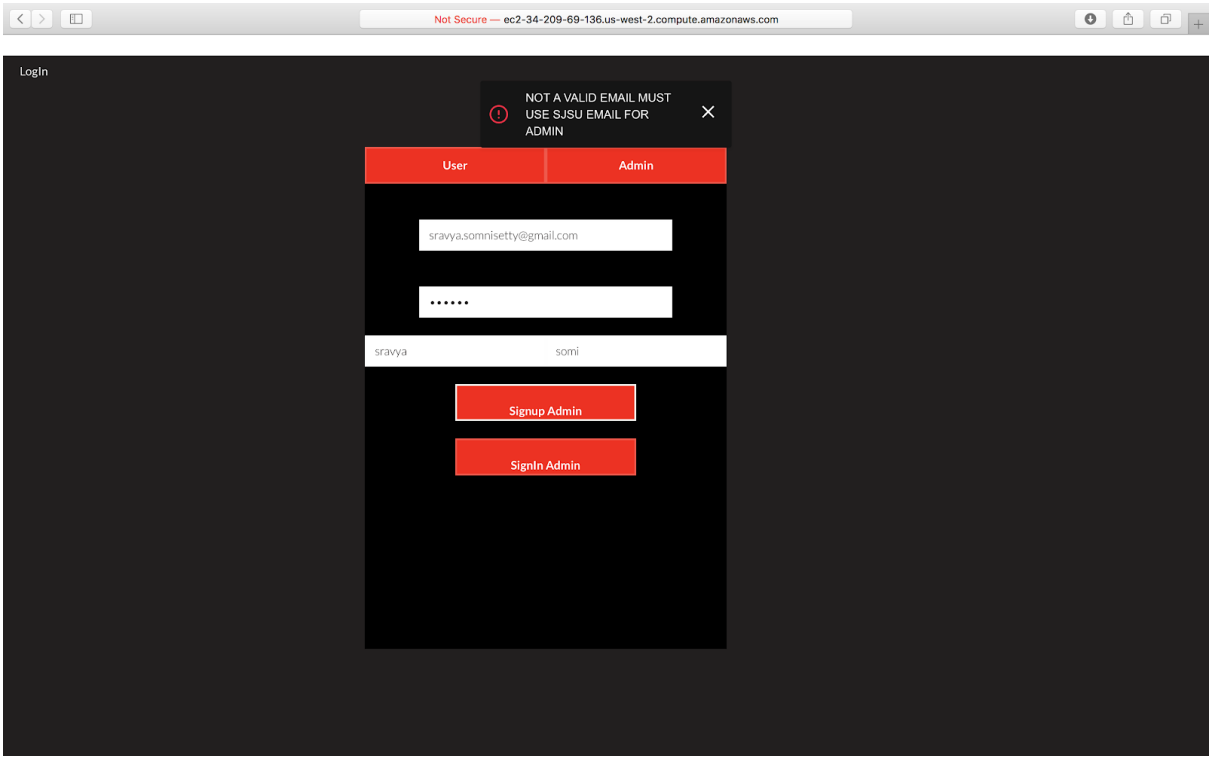
1. Front End: (HTML, CSS, Java script, Bootstrap, React JS)
    a. React JS for building User Interface.
    b. Redux for Application State Management.
    c. Firebase for OAuth using Google and Facebook for the user.
2. Back End:(Java 10, Spring Restful API)
    a. Java Spring Boot and backend data is accessed as Restful Web Service.
    b. Maven as a dependency management tool.
    c. Log4j is used for logging the information throughout the flow
    d. Spring AOP for writing aspects in the backend.
    e. Spring Security is used for authorization and authentication
    f. Spring Data ElasticSearch is used for ElasticSearch implementation in the project.
    g. Spring Data JPA as ORM for Persisting data to MySQL.
    h. Junit, mockito and power mockito for unit testing
3. Data Base
    a. Amazon cloud RDS MySQL.
    b. Amazon cloud Elastic Search for ElasticSearch implementation.
4. Cloud Service
    a. Amazon Cloud EC2 is used for deploying frontend and backend modules.
    b. Amazon Cloud Elastic Search for Elastic cloud.

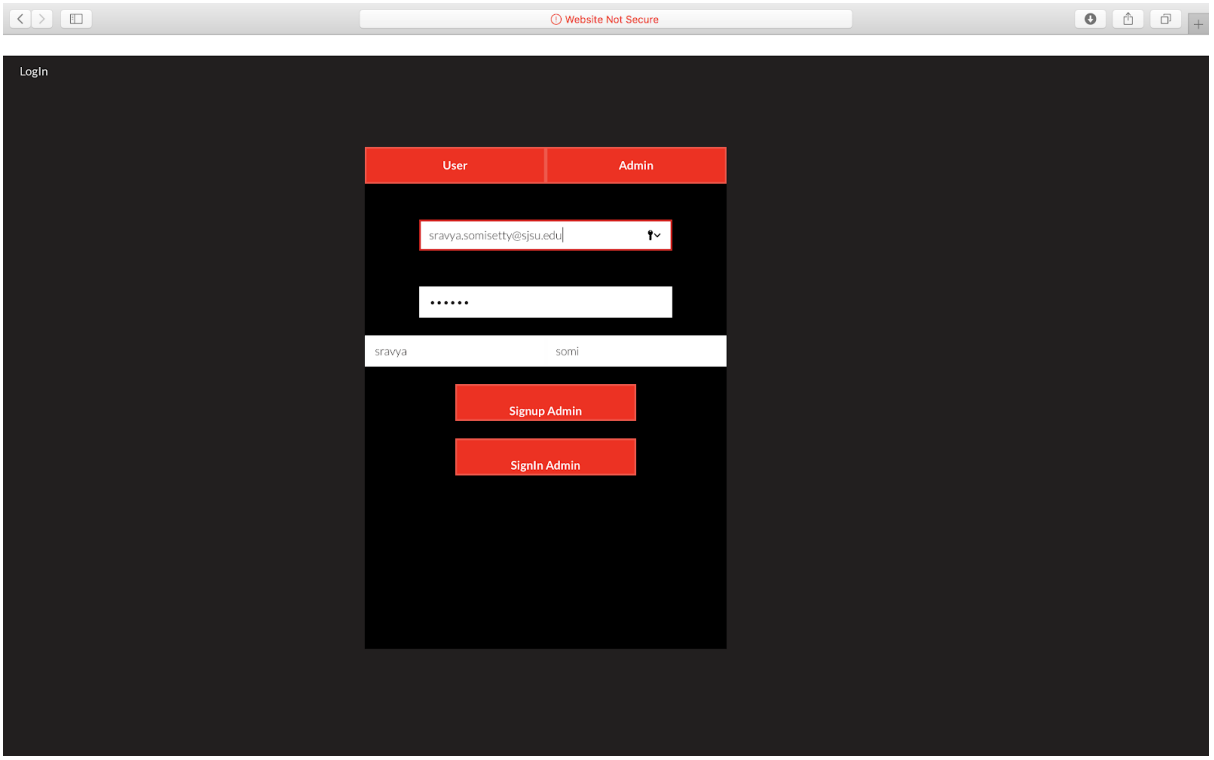
5. Description of Features with Screenshots

1. admin login/signup:

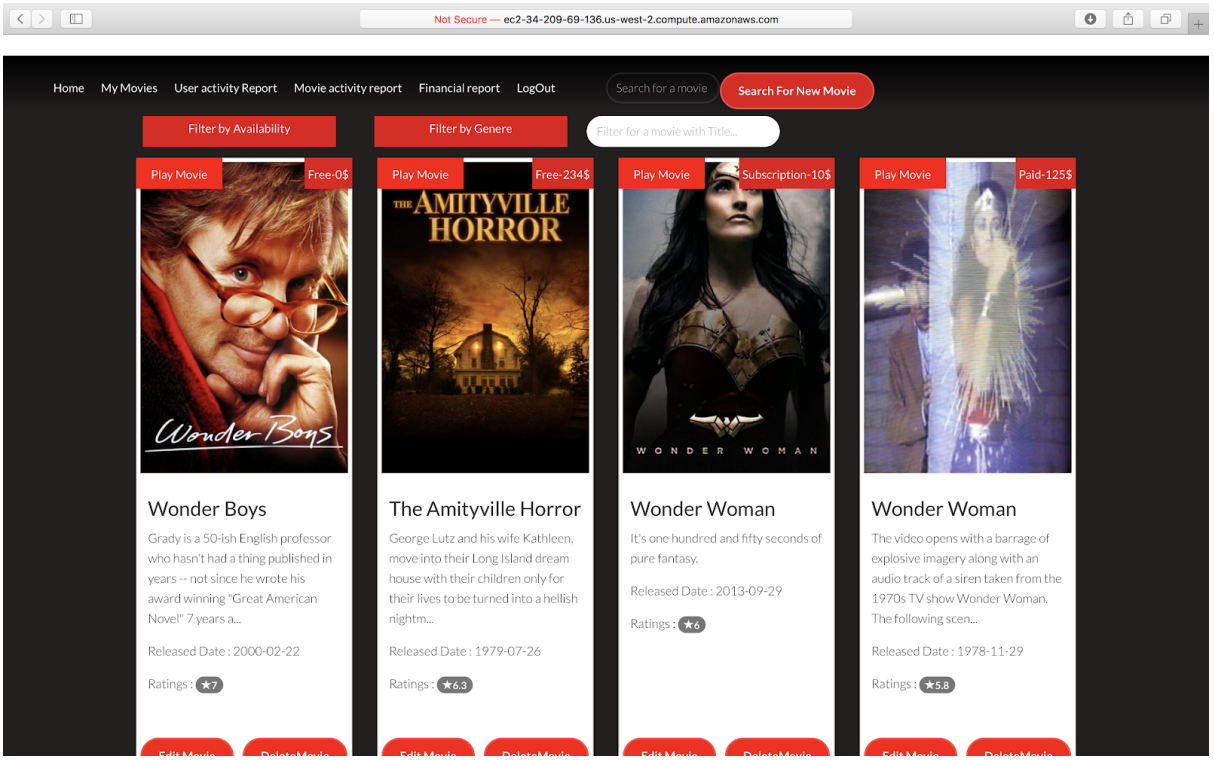
Following is the page for Admin login and login is restricted for admins with sjsu IDs.

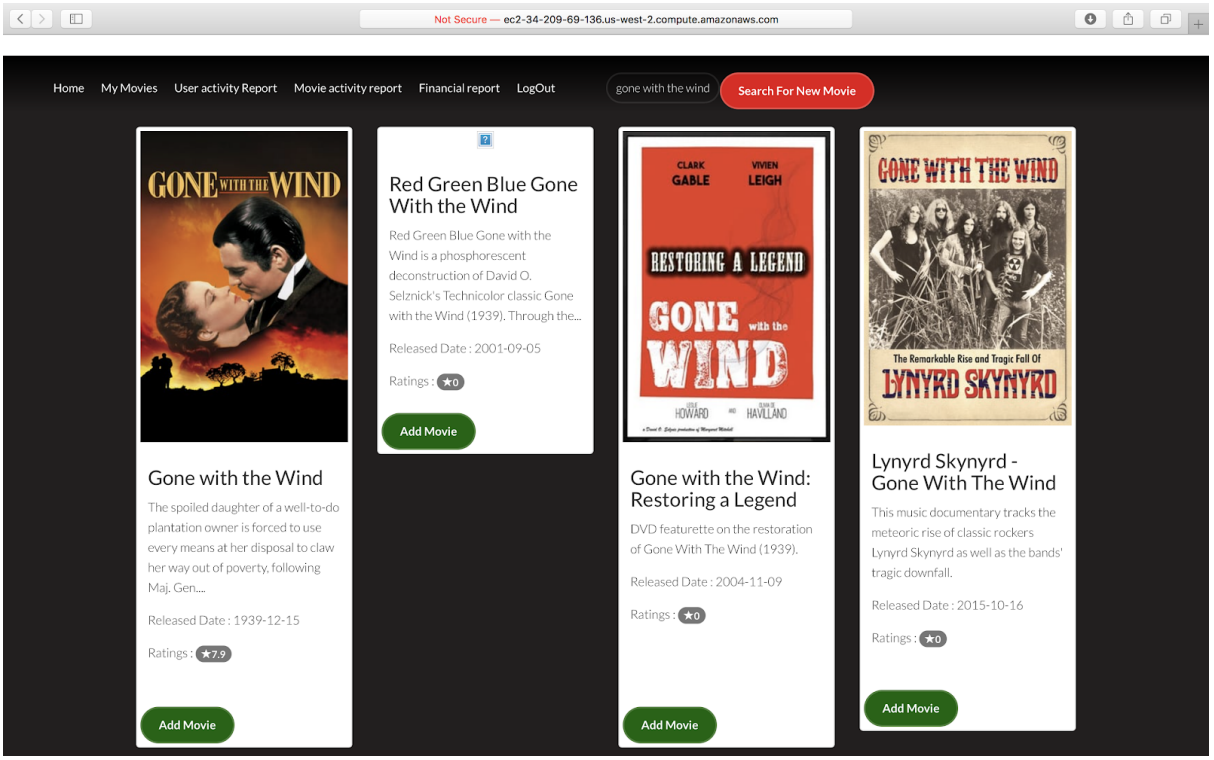

Search or enter website name
LogIn
User
Admin
Email Please
Password Please
SignIn User
SignUp User
Sign in with Google
Sign in with Facebook


Not Secure — ec2-34-209-69-136.us-west-2.compute.amazonaws.com
LogIn
NOT A VALID EMAIL MUST USE SJSU EMAIL FOR ADMIN
User
Admin
sravya.somnisetty@gmail.com
••••••
sravya
somi
Signup Admin
SignIn Admin

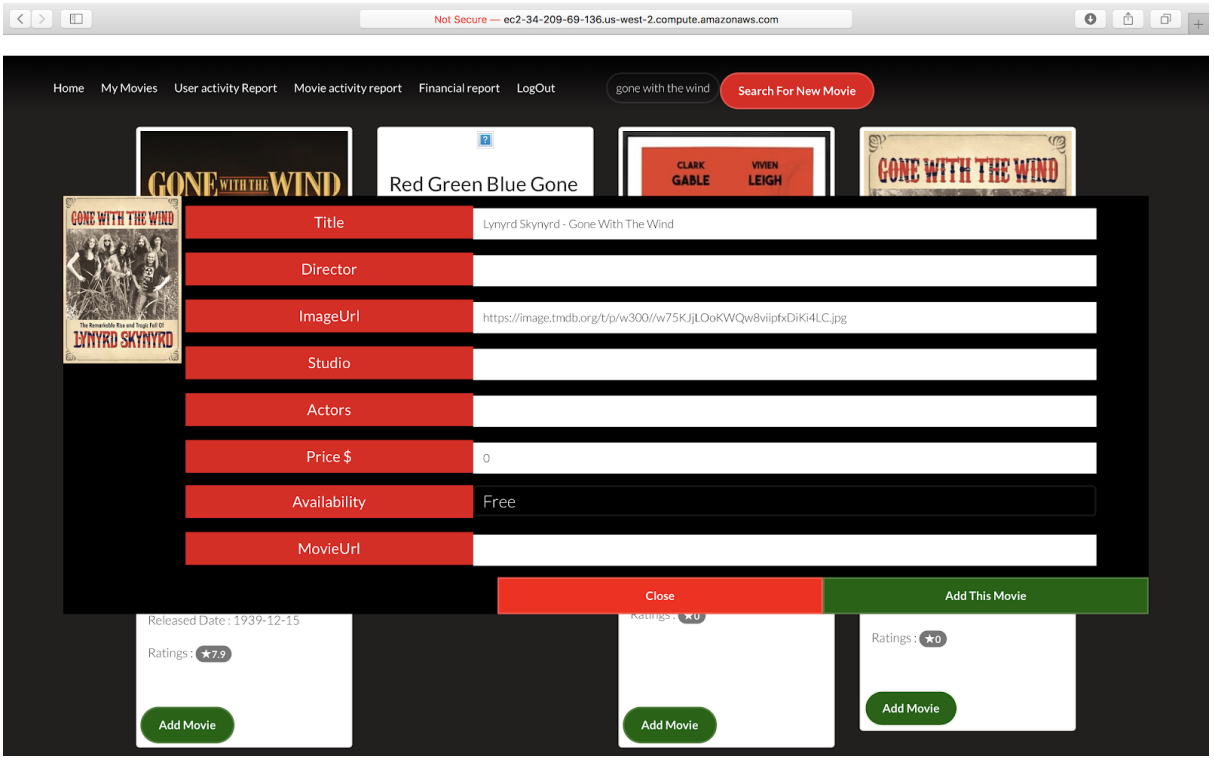


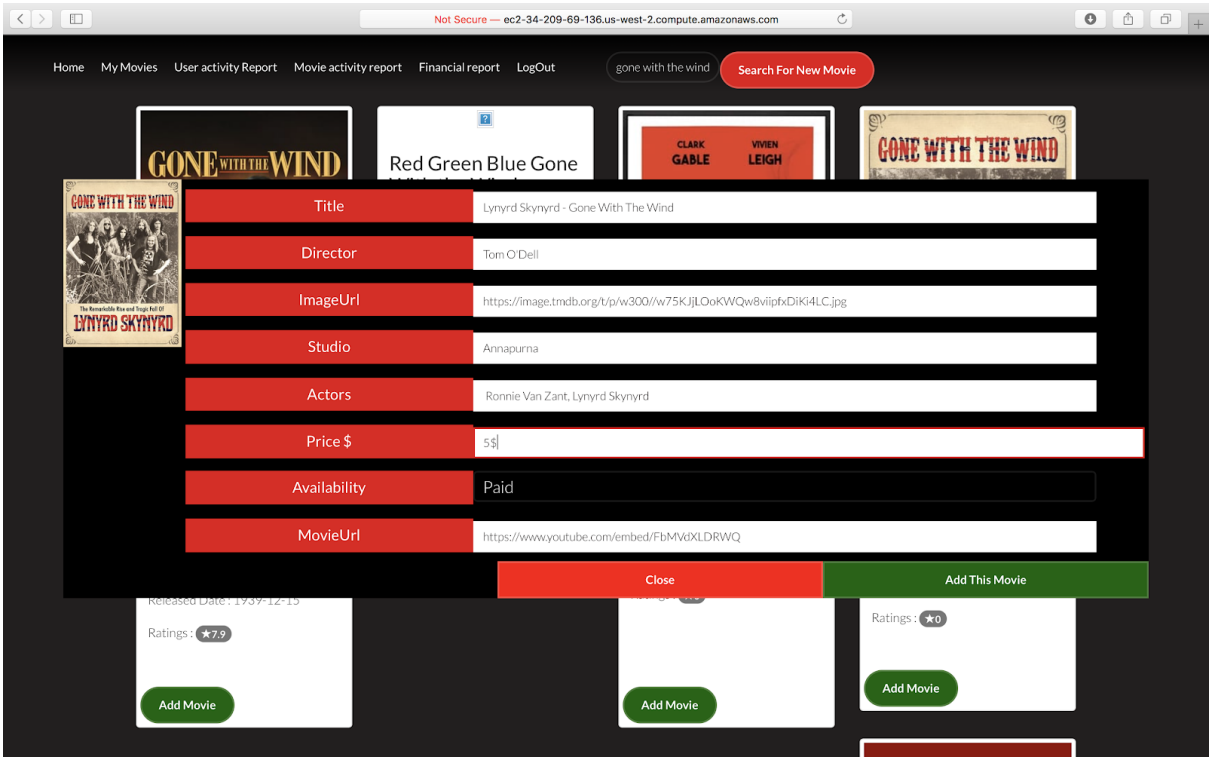
2. Following is the screen for my movies page.



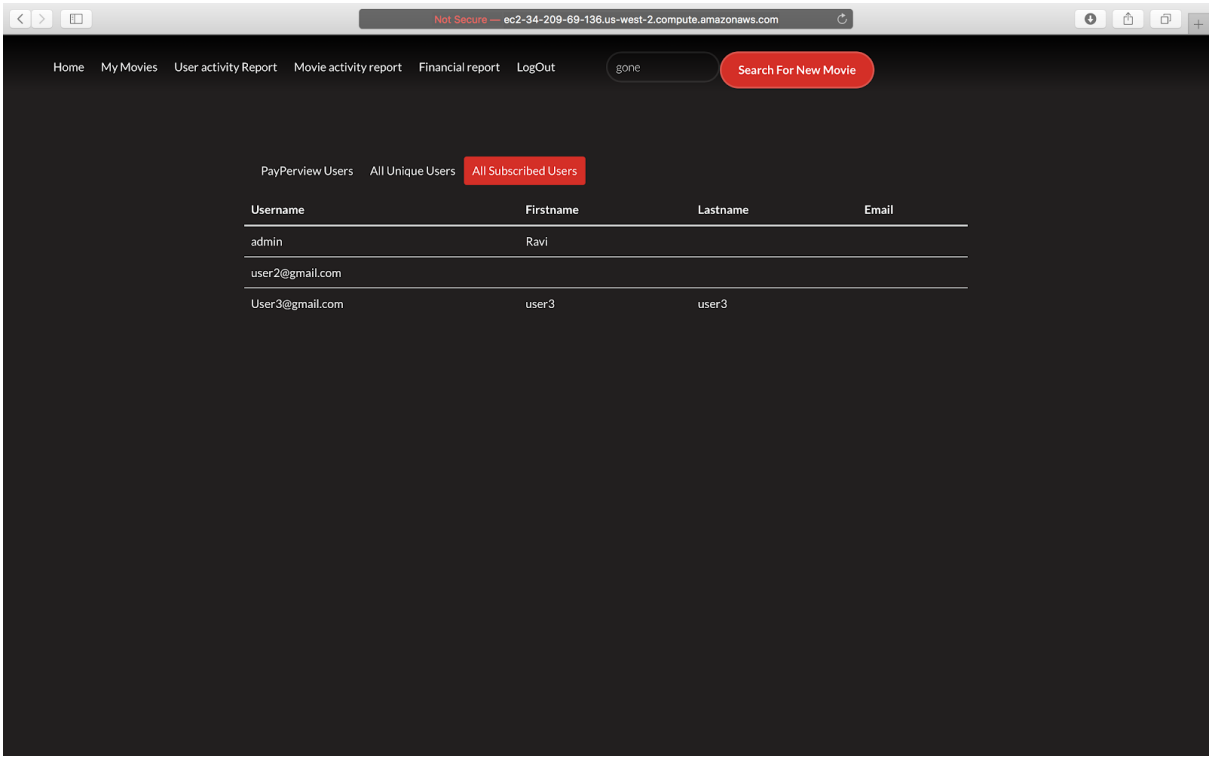


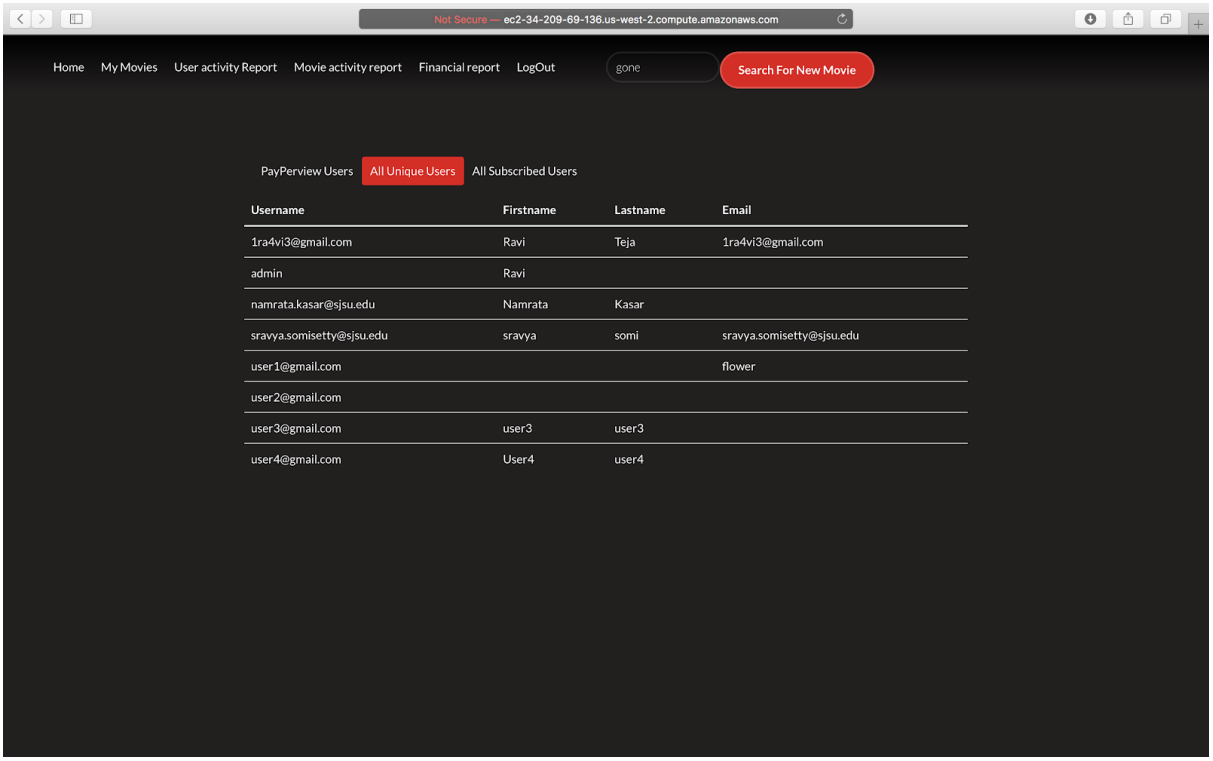
3. Following is the screen for adding and editing movie details for admin:

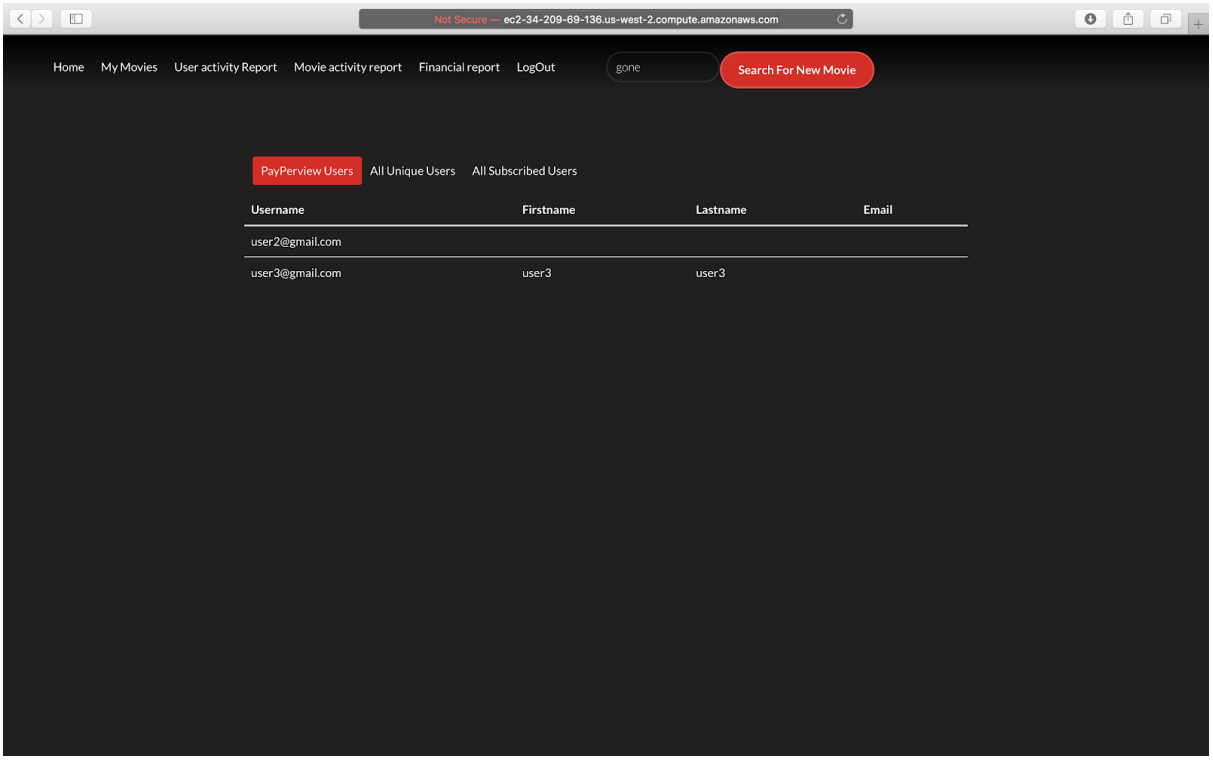




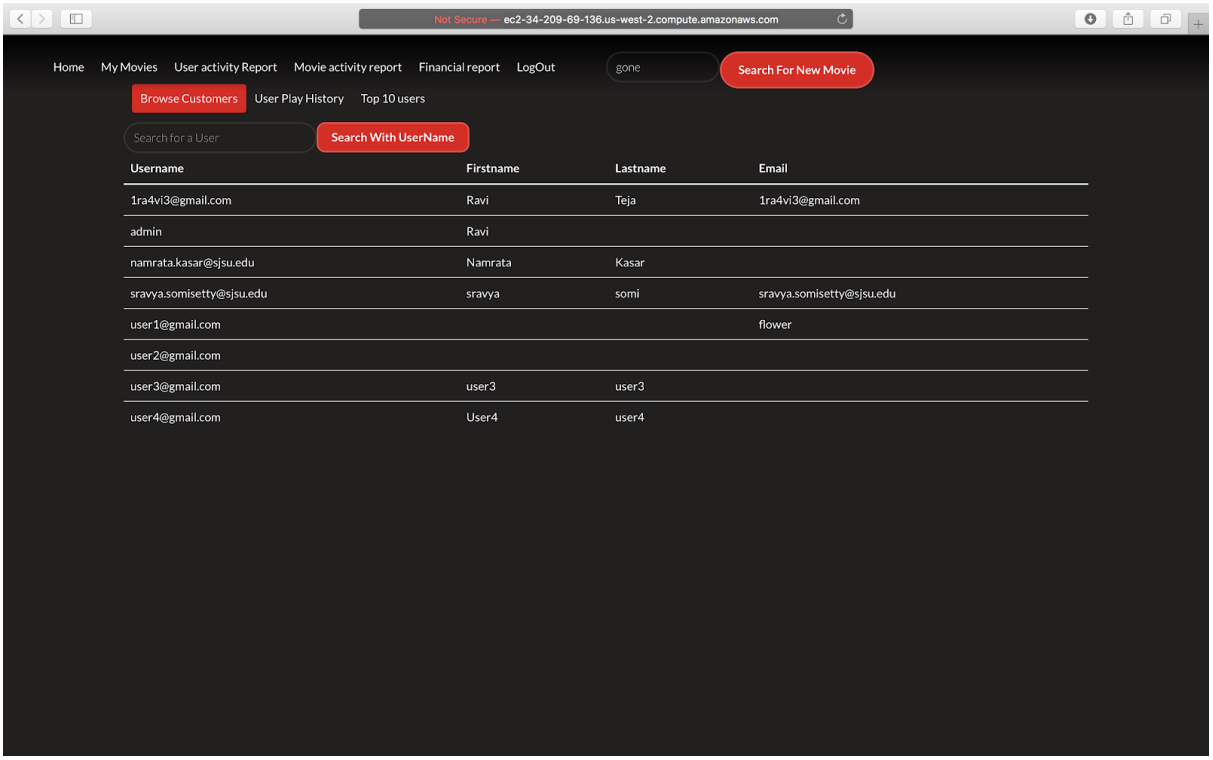
4. Following are the screens financial report screen for Admin


Not Secure — ec2-34-209-69-136.us-west-2.compute.amazonaws.com
Home
My Movies
User activity Report
Movie activity report
Financial report
LogOut
gone
Search For New Movie
PayPerview Users
All Unique Users
All Subscribed Users
Username
Firstname
Lastname
Email
admin
Ravi
user2@gmail.com
User3@gmail.com
user3
user3

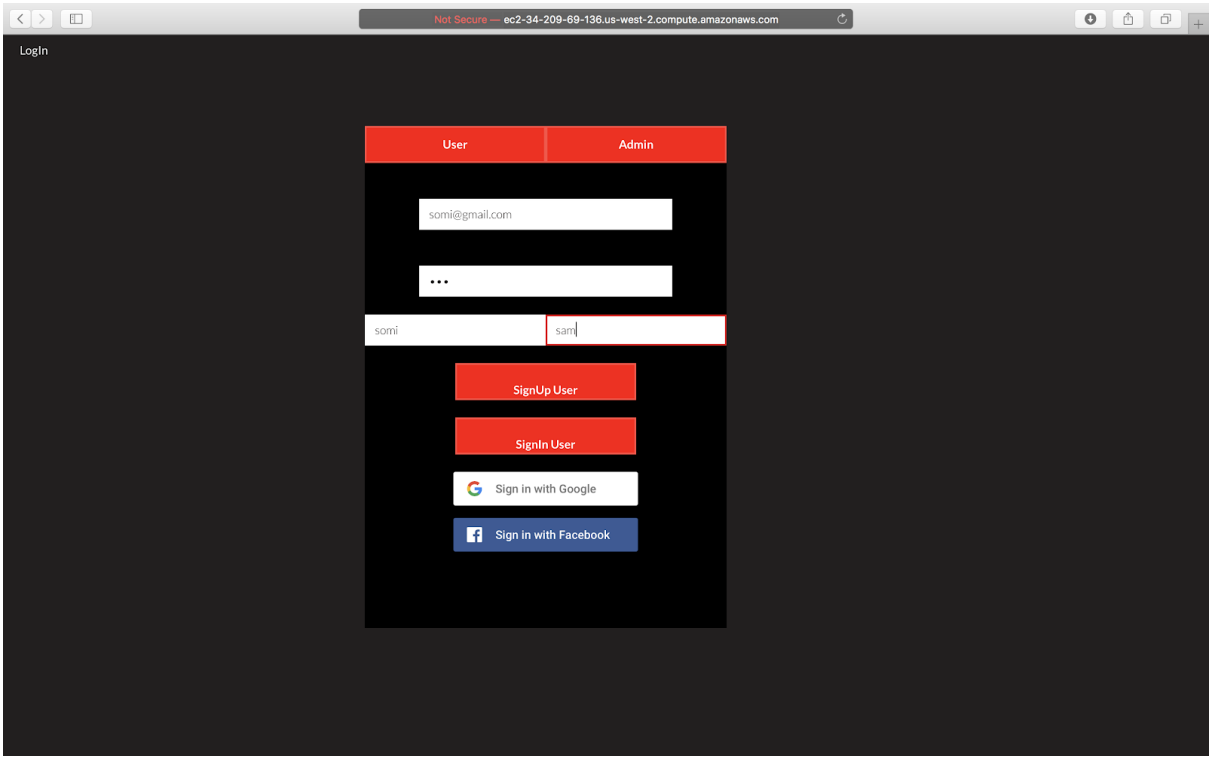

Not Secure — ec2-34-209-69-136.us-west-2.compute.amazonaws.com
Home
My Movies
User activity Report
Movie activity report
Financial report
LogOut
gone
Search For New Movie
PayPerview Users
All Unique Users
All Subscribed Users
Username
Firstname
Lastname
Email
1ra4vi3@gmail.com
Ravi
Teja
1ra4vi3@gmail.com
admin
Ravi
namrata.kasar@sjsu.edu
Namrata
Kasar
sravya.somisetty@sjsu.edu
sravya
somi
sravya.somisetty@sjsu.edu
user1@gmail.com
flower
user2@gmail.com
user3@gmail.com
user3
user3
user4@gmail.com
User4
user4



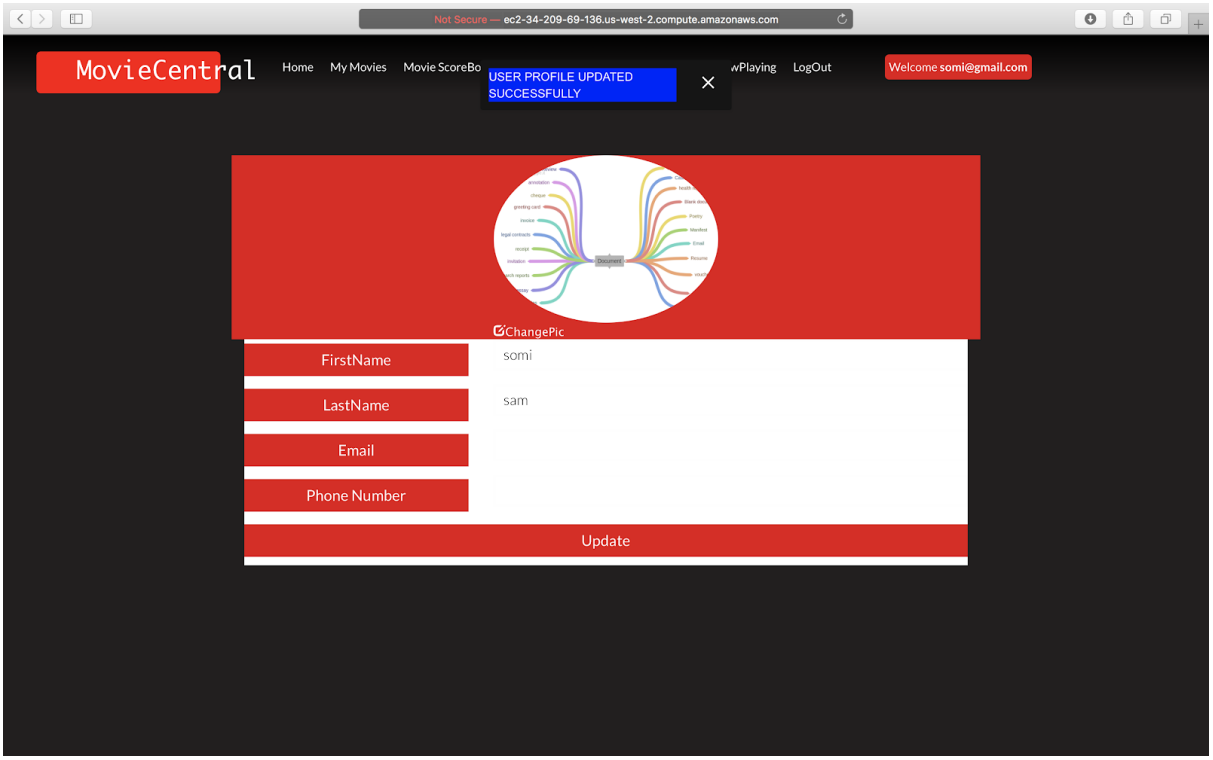
5. Following is the screen for user activity report for admin.



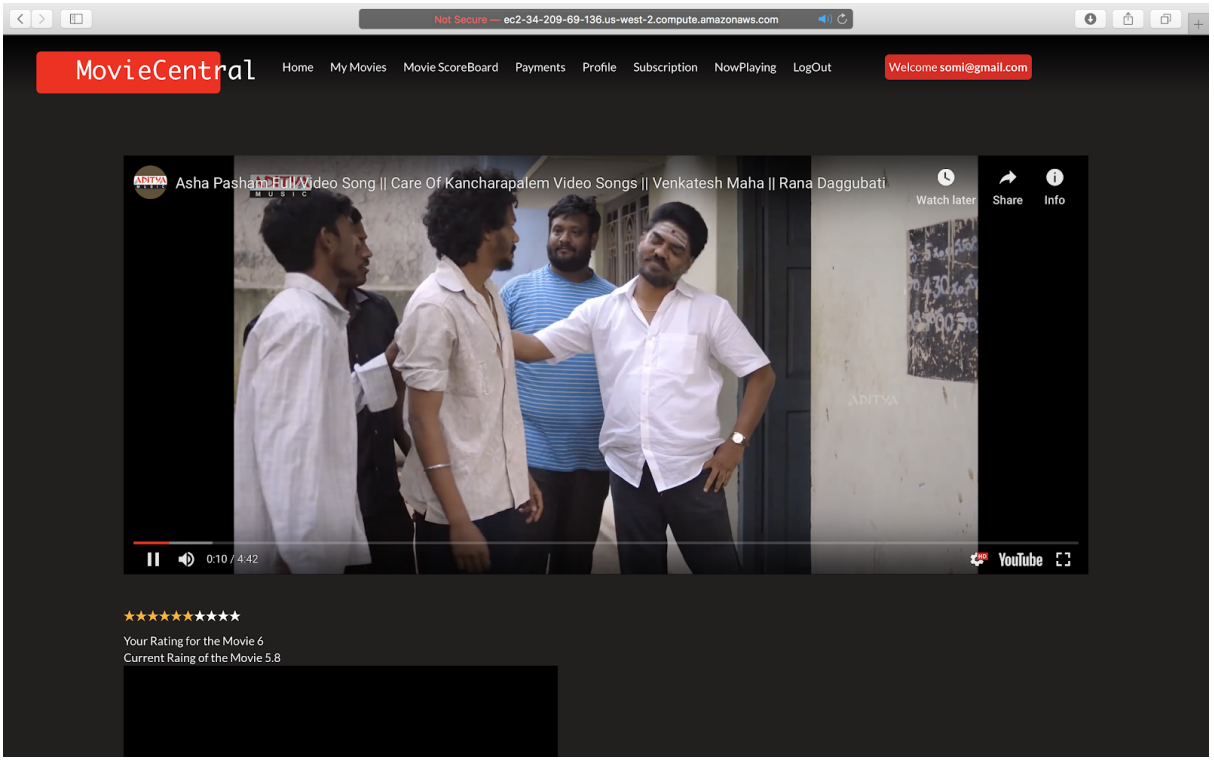
6. Following is the screen for user login.



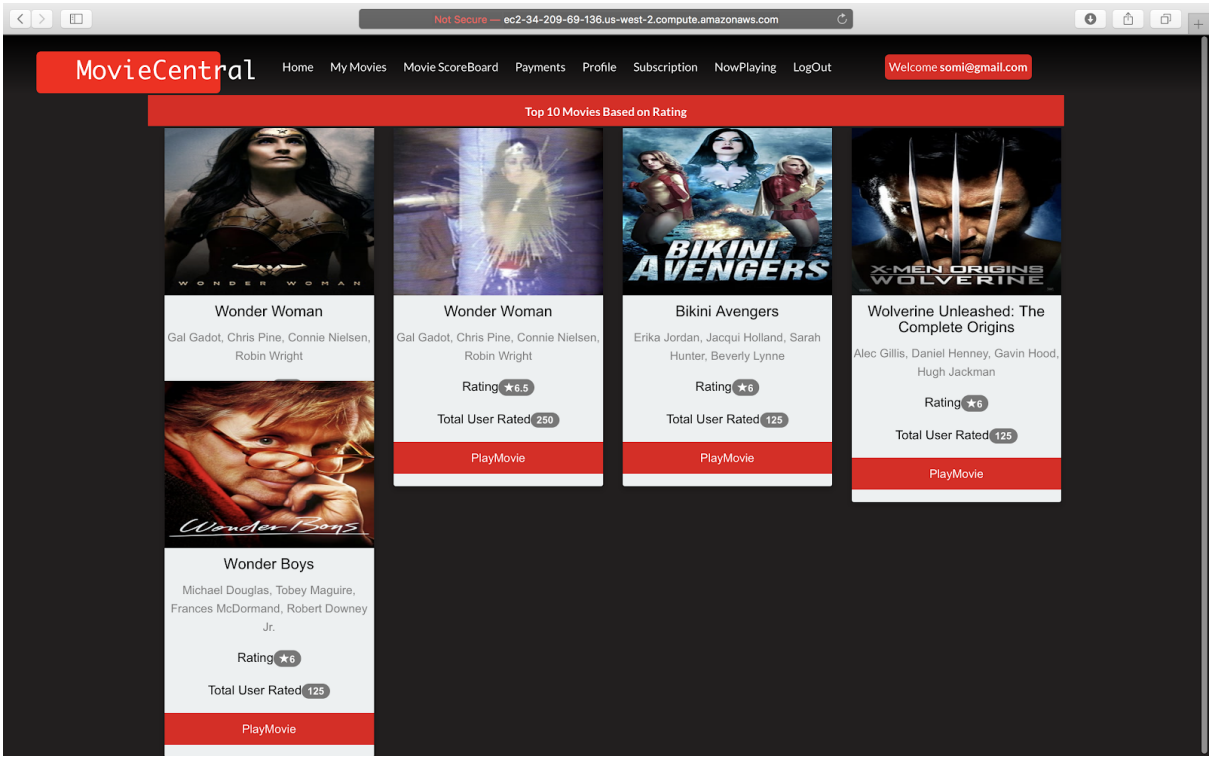
7. Following is the screen for user profile where he can edit the details.



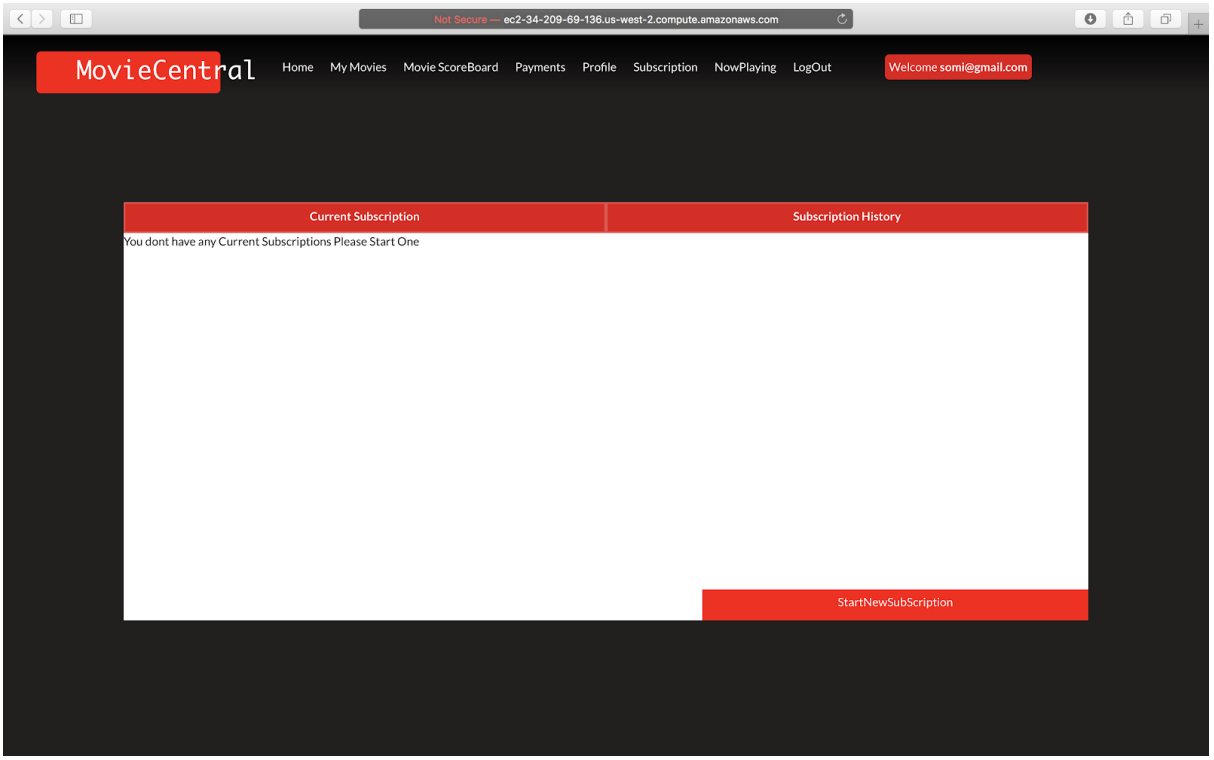
8. Following is the screen where user can watch the movie.

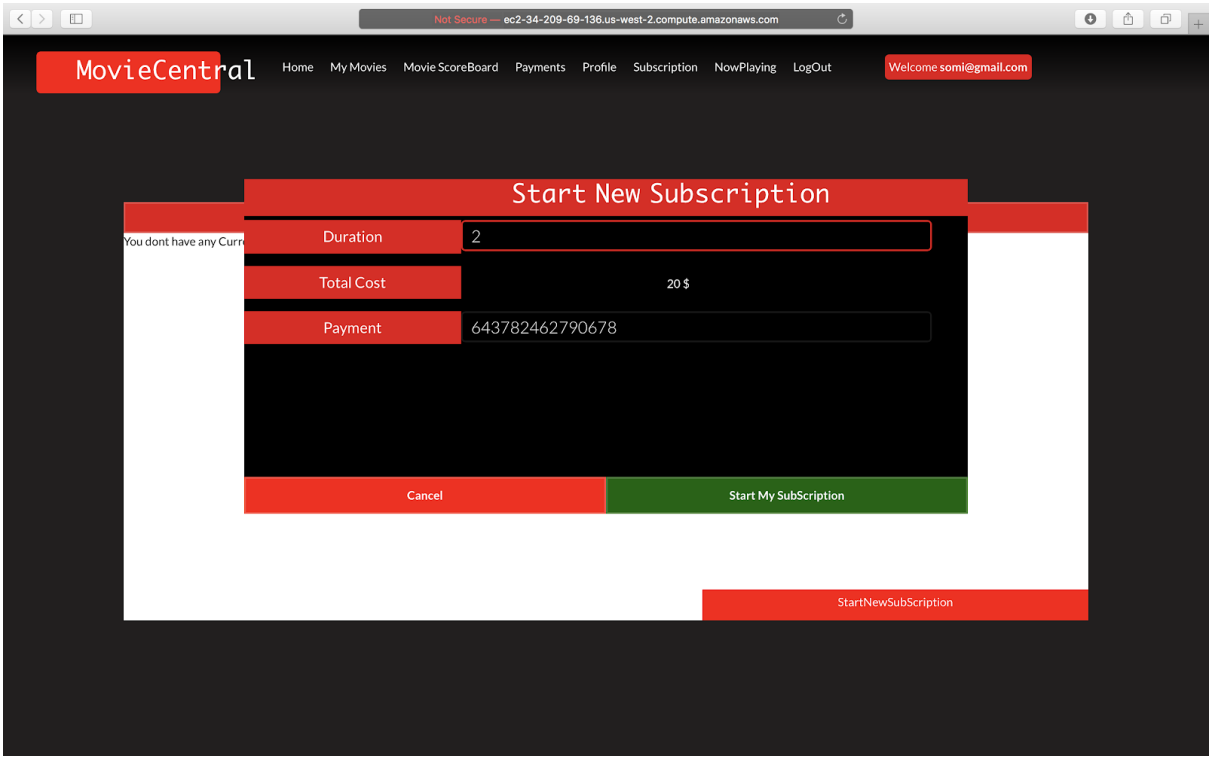


9. Following is the scoreboard for user where he can see top 10 movies.

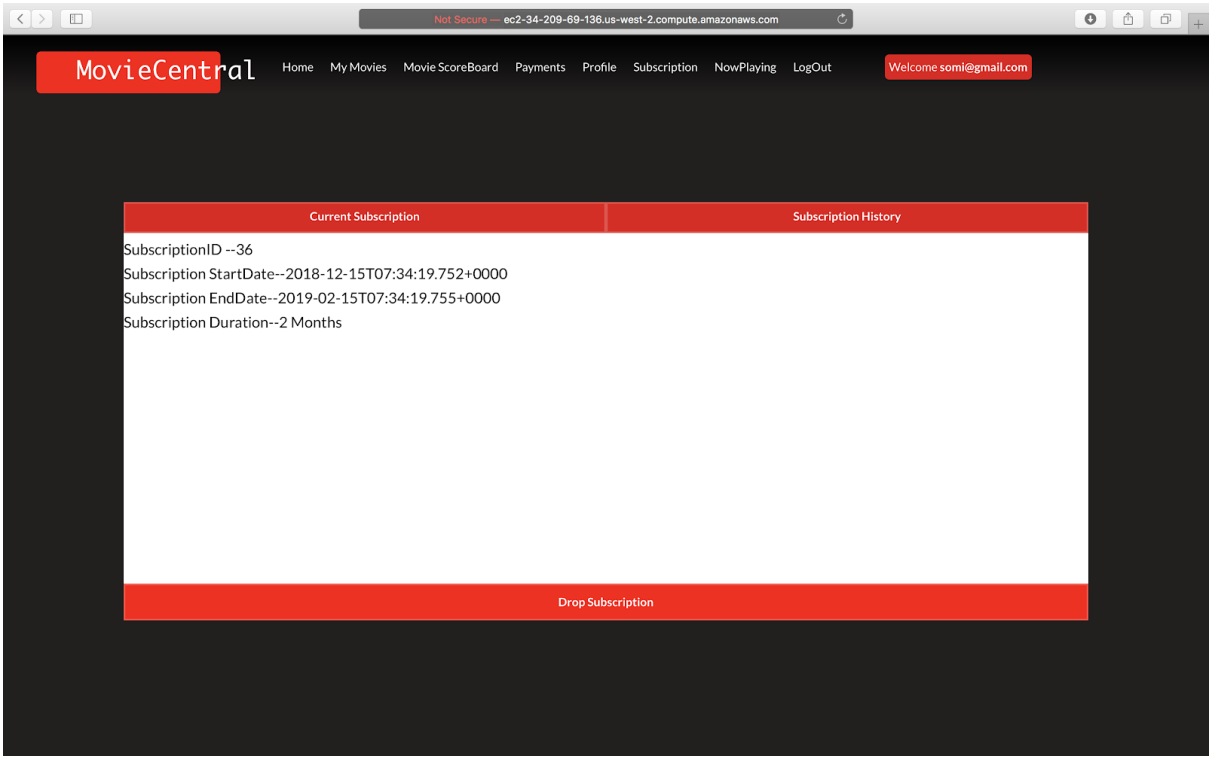


10. Following are the screens for subscription of user where he can subscribe himself.

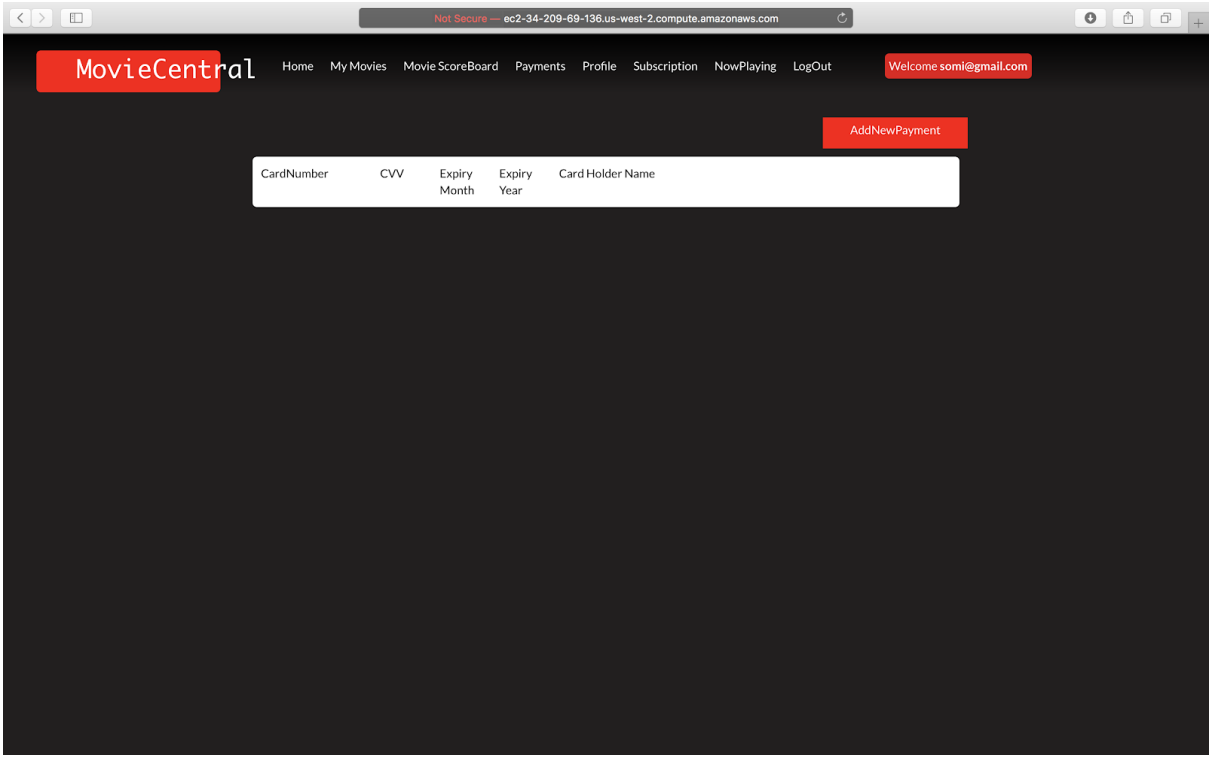


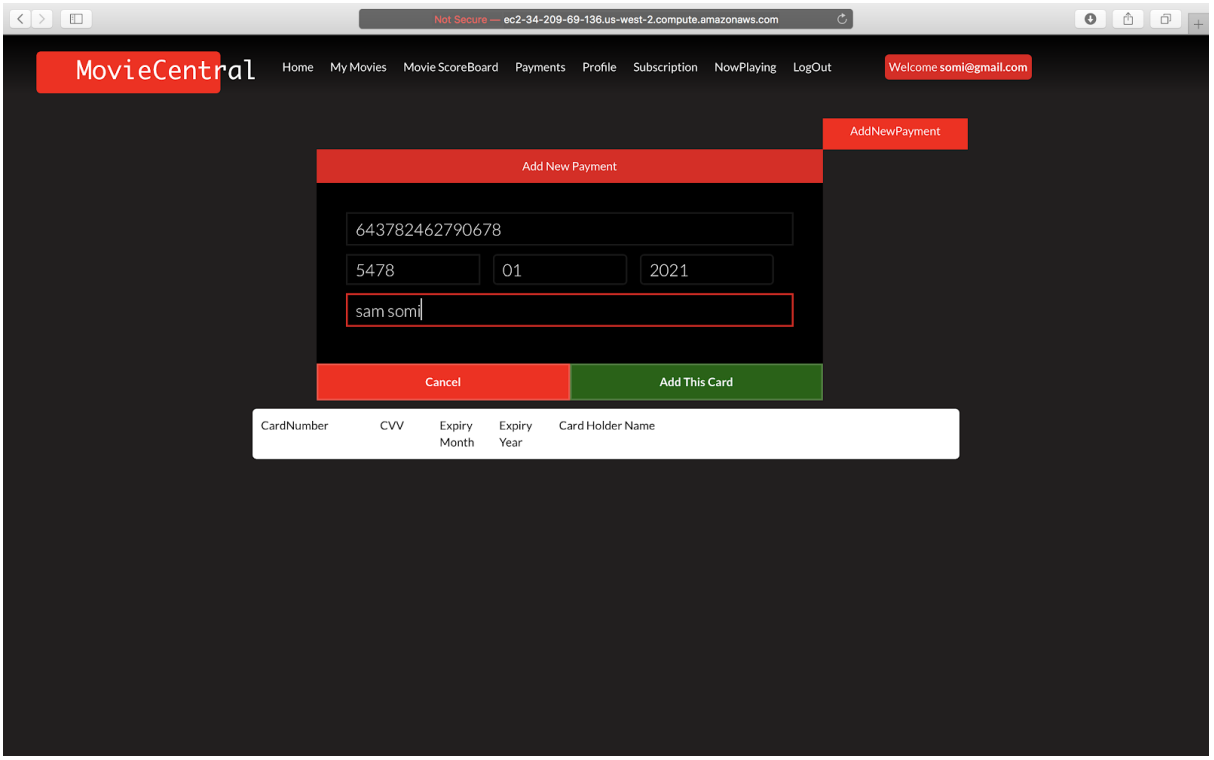


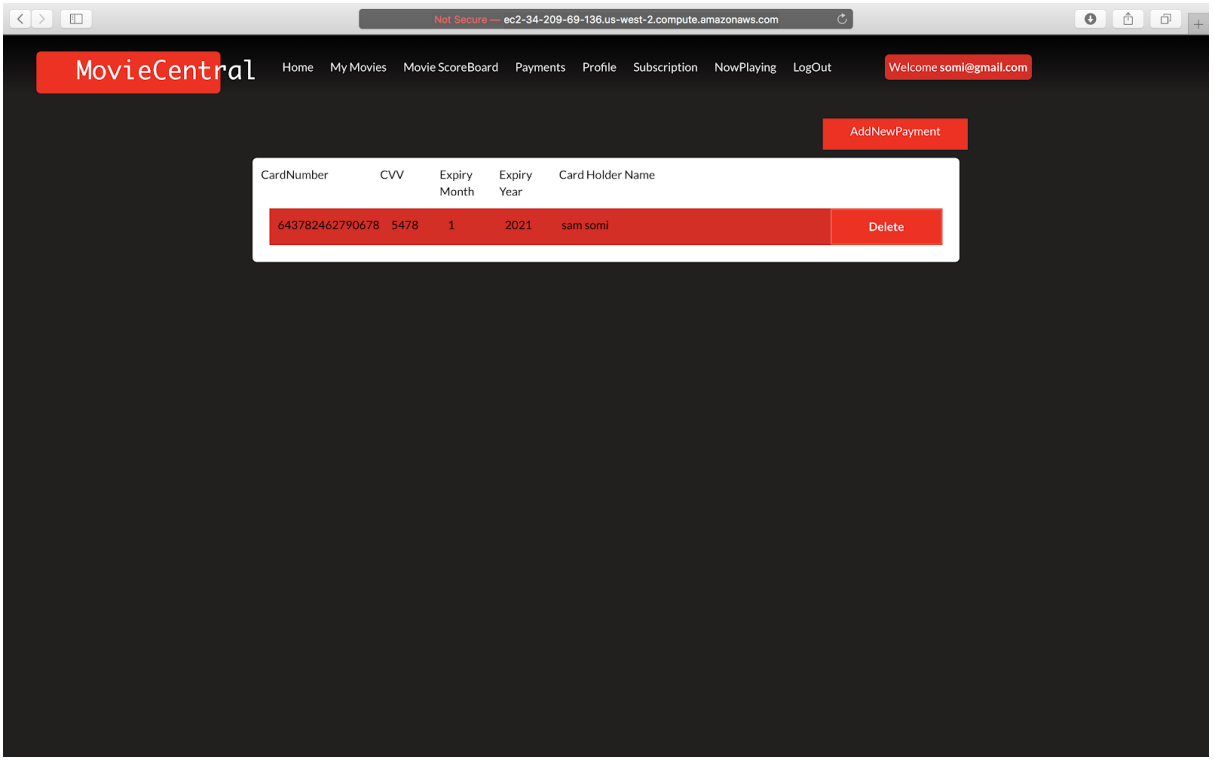
11. Following is the screen to see user's subscription details.



12. Payment screens for user.







6. Testing plan executed and results

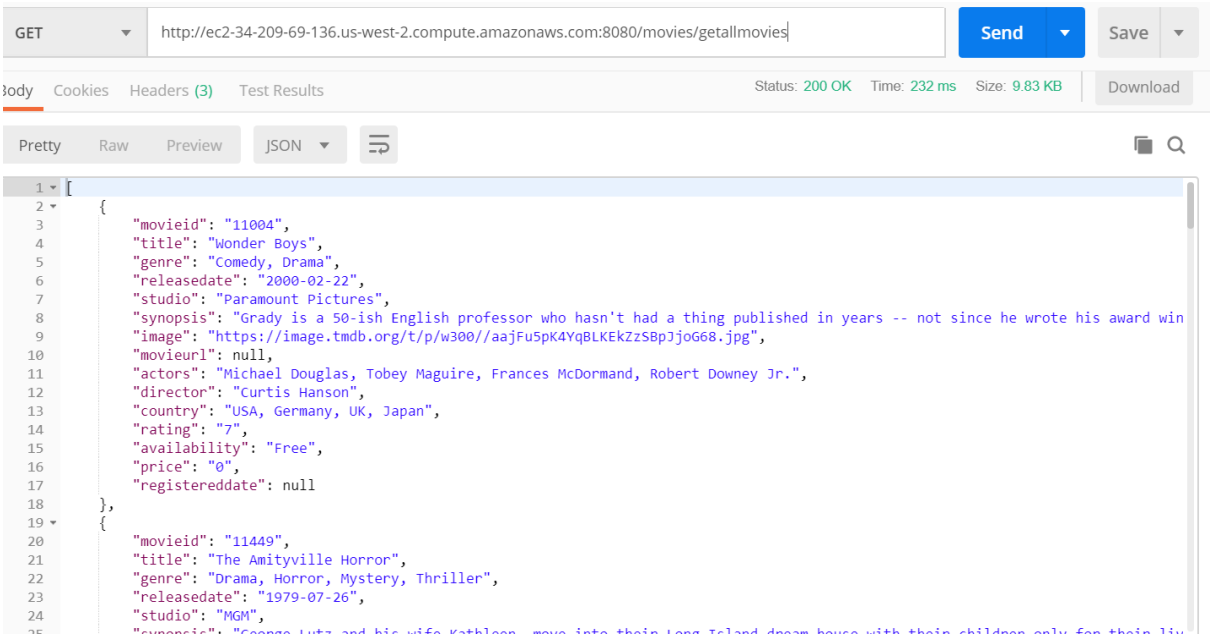
We have used junit, mockito, power mock for unit testing the classes and we have used mockito for mocking the dependent classes.

Power mockito is used for mocking private methods in the classes. Example of the test case is given below:

```java
public class UserControllerTest {

    @Test
    public void testuserVerification() {

        UserController usercontroller =new UserController();
        User user =new User();
        user.setUsername("User1@gmail.com");
        user.setPassword("user");
        User res=usercontroller.userSignin(user,null);
        assertEquals(res.isEnable(), true);
    }

}
```

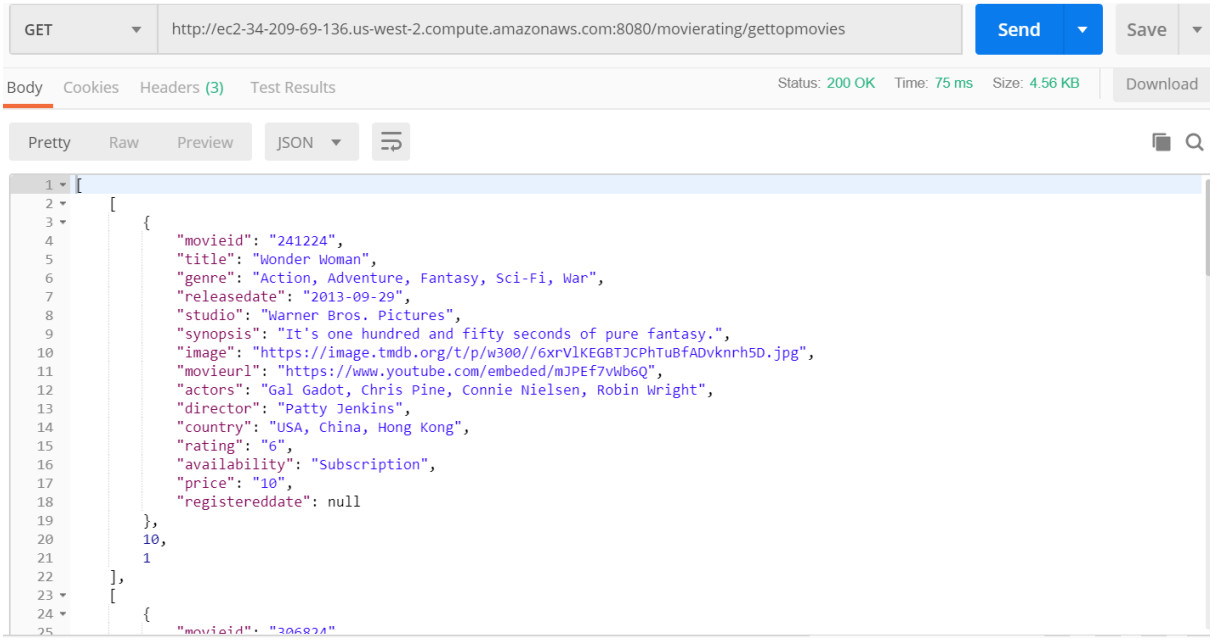
Expected response is checked on Postman for the particular REST api urls. Following screenshots show few of the responses we got for the REST APIs included in the project.

1. Get all the movies:
   API url:
   http://ec2-34-209-69-136.us-west-2.compute.amazonaws.com:8080/movies/getallmovies



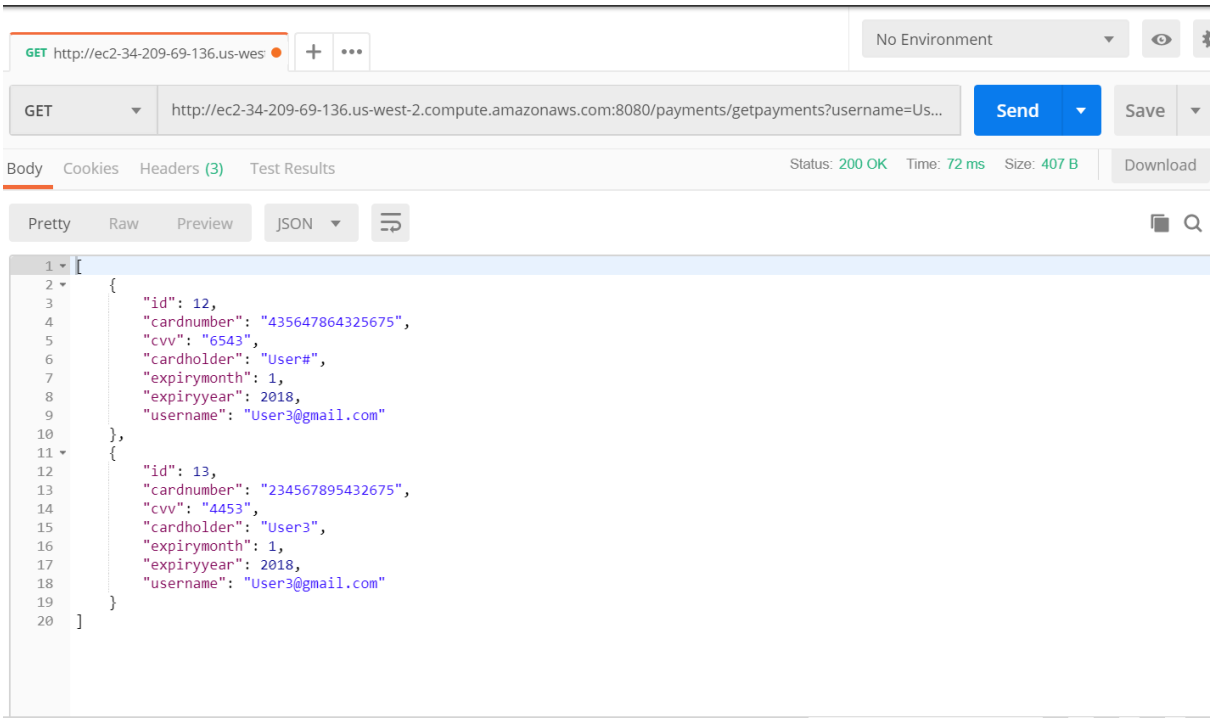
2. Get top 10 movies:
   API url :
   http://ec2-34-209-69-136.us-west-2.compute.amazonaws.com:8080/movierating/gettopmovies

   Expected: Top 10 movies should be displayed.
   Actual: List of movies with details (upto 10) is displayed as follows:



3. Get all payment details for user “User3@gmail.com”
   API url :
   http://ec2-34-209-69-136.us-west-2.compute.amazonaws.com:8080/payments/getpayments?username=User3@gmail.com

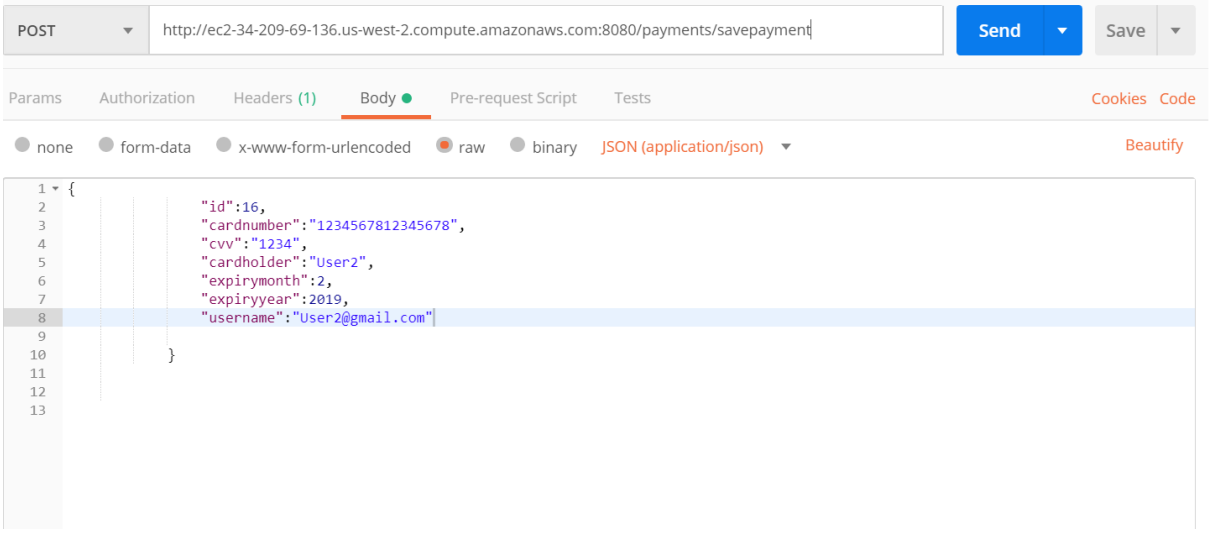
   Expected : All payment details (card details) of user should be displayed.
   Actual : List of all payments of that user as per the database is seen as follows:

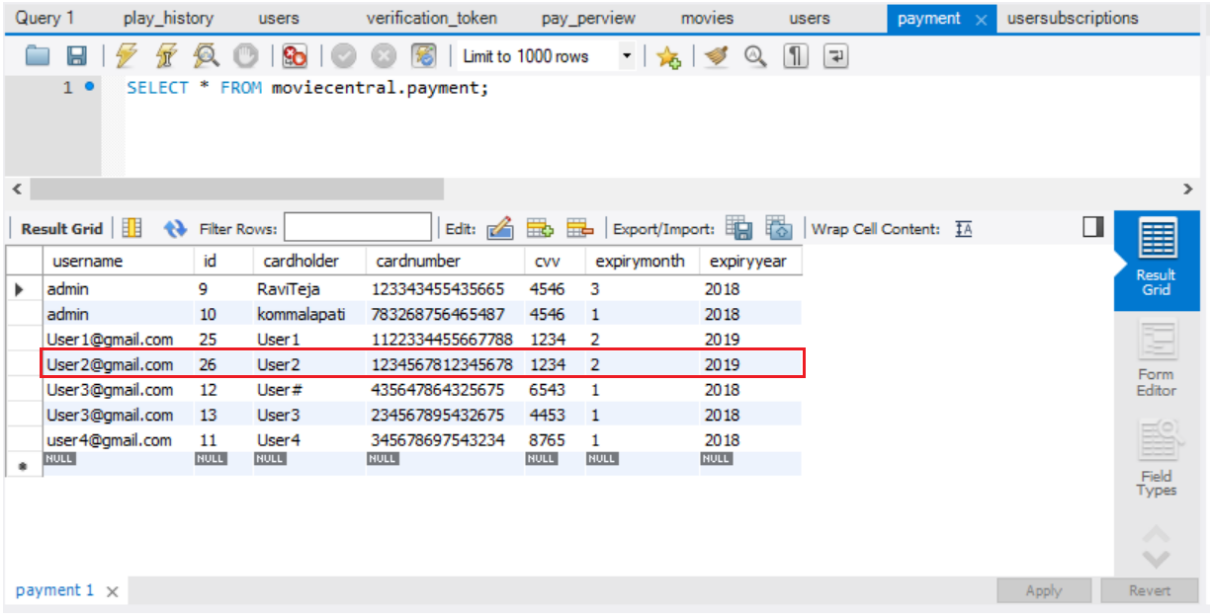


4. Saving payment details for user “User1@gmail.com”.

API url:
http://ec2-34-209-69-136.us-west-2.compute.amazonaws.com:8080/payments/savepayment

Expected: Payment details should be saved and payment object should be displayed.
Actual: Payment details are saved and payment object is displayed as follows:





7. Lessons learned and possible Future Work

- We learned about Aspect Oriented Programming and how it can implemented to overcome cross cutting concerns like Logging and Exception Handling.
- We have learnt about Spring Security, how it can be implemented to protect the API by using authentication and Authorization.
- We have implemented hibernate JPA for database management where we learnt about ORM mapping and table inheritance concepts and how to use the named queries.
- We learnt about implementation of CORS for project level not for method level.

- We have used annotation throughout the project to make efficient use of declarative programming.
- We have implemented elastic search for recommendations and making large scale data queries(full text search).
- As future scope, we can add TV shows and we can stream live videos as well.