



**SOEN 6441 – Fall 2021**

---

# **SOFTWARE ARCHITECTURE DOCUMENT**

---

## **News Application System**

### **Designed By:**

Visnunathan Chidambaranathan (40230157)

Nisha Bhatia (40220958)

### **Faculty Coach:**

Dr. Constantinos Constantinides

## Table of Contents

<b>1. INTRODUCTION.....</b>	<b>3</b>
<b>1.1 PURPOSE.....</b>	<b>3</b>
<b>1.1.1 PROBLEM DEFINITION.....</b>	<b>3</b>
<b>1.1.2 OBJECTIVES.....</b>	<b>4</b>
<b>1.2 NON-FUNCTIONAL REQUIREMENTS.....</b>	<b>4</b>
<b>1.3 TECHNOLOGIES USED.....</b>	<b>5</b>
<b>2. ARCHITECTURE OVERVIEW.....</b>	<b>6</b>
<b>2.1 LOGICAL VIEW.....</b>	<b>7</b>
<b>2.2 PROCESS VIEW.....</b>	<b>9</b>
<b>2.3 DEPLOYMENT VIEW.....</b>	<b>10</b>
<b>2.4 USE CASE DIAGRAM.....</b>	<b>10</b>
<b>3. USER INTERFACE DESIGN.....</b>	<b>12</b>
<b>3.1 LOGIN PAGE.....</b>	<b>12</b>
<b>3.2 REGISTRATION PAGE.....</b>	<b>13</b>
<b>3.3 REVIEWER VIEW.....</b>	<b>13</b>
<b>3.4 USER VIEW.....</b>	<b>14</b>
<b>3.5 REVIEW DATA LIST.....</b>	<b>14</b>
<b>3.6 SUBSCRIBING REVIEWER AND SUBSCRIBED DATA LIST.....</b>	<b>15</b>
<b>4. TESTING.....</b>	<b>16</b>
<b>4.1 DATABASE.....</b>	<b>16</b>
<b>4.2 WEB SERVER.....</b>	<b>17</b>
<b>4.3 WEB APPLICATION.....</b>	<b>17</b>
<b>5.CODE REFACTORING.....</b>	<b>19</b>
<b>6.GITHUB REPOSITORY.....</b>	<b>24</b>
<b>7. REFERENCE.....</b>	<b>25</b>

# 1. INTRODUCTION

---

In this project we are entitled to implement various design patterns by fetching the data from the APIs, storing the data into database and further showing it to the viewers. The purpose of this document is to present our application's architectural layout and describe the relationships between the integrated objects and the processes that are included in it.

The objective of this application is to help users read about their subscribed articles, reviews, and other items that are posted on their feed by receiving notifications of what they posted. To implement this, we have used the Observer design pattern to map the observers and the listeners, and the Table Data Gateway, Data Source architectural pattern to map the data we fetched from the API. We have fetched the data from 2 APIs namely Movie Review API and Article Search API. This document will address the background for this project, and the architecturally significant functional requirements.

The future scope of this project is that we can extend the functionality by using the data of Article search API and extend our application for further requirements. The intention of this document is to help the development team to determine how the system will be structured at the highest level.

## 1.1 PURPOSE

---

### 1.1.1 PROBLEM DEFINITION

---

To design a system that can read API/JSON data from The New York Times and produce results to application:

- Data collection from The New York Times.

- Displaying it in the User Interface.
  - Designing Observer Designer Pattern in the existing system.
- 

### 1.1.2 OBJECTIVES

---

Our system aims to produce a web application which will represent the Observer design pattern in the data which was obtained from the API (The New York Times). The Reviewers will be given a separate layout to post their reviews regarding the movie. The subscribers will be allowed to subscribe to their own list of reviewers and the post which was posted by the reviewer will be notified to them. Through this document, the architecture of the system will be described, as a way that compliments the code and describes what the code itself wouldn't do. This document is intended to describe the architectural decisions which have been made on the system.

## 1.2 NON-FUNCTIONAL REQUIREMENTS

---

1. **High performance:** The system must be able to receive a big number of API data and be able to process them and store it on the database each second.
2. **User friendly:** the final users will be the users and will be easily accessible to them to view all the data's retrieved from API.
3. **Security:** all the components must be totally secured, in order to prevent leaks and intrusions that could imply physical security issues inside the stores and data manipulation to harm the company. (Prepared statements are used to stop SQL injection.)
4. **Human errors:** humans are the #1 source of involuntary (or voluntary) cause of problems in the systems. The system should always check the user input and, in general, any instruction.
5. **Maturity:** the system must run tests every time that a change is made, and new tests have to be built for new features. Both unit and inter-module tests should be done.

6. **Changeability:** everything is very likely to change, so the system must be able to handle any kind of changes in features.

## 1.3 TECHNOLOGIES USED

---

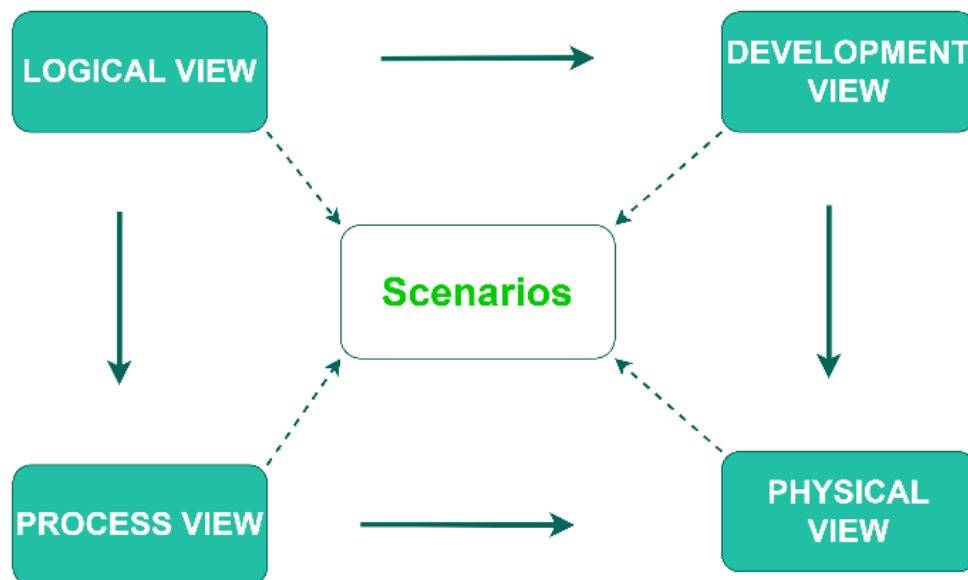
- JAVA
- JSP
- SERVLET
- MYSQL
- HTML
- CSS
- JAVASCRIPT
- SELENIUM

## 2. ARCHITECTURE OVERVIEW

---

This document is the first approach to present the information of this project in a structured fashion and discuss its architecture.

We are going to explain our application functionality with the help of “4+1 architectural view model” which includes Logical View, Process View, Development View, Physical View, and the scenarios will illustrate the entire architecture of our system. Each diagrams represents an in-depth detailed explanation for the system that has been developed.

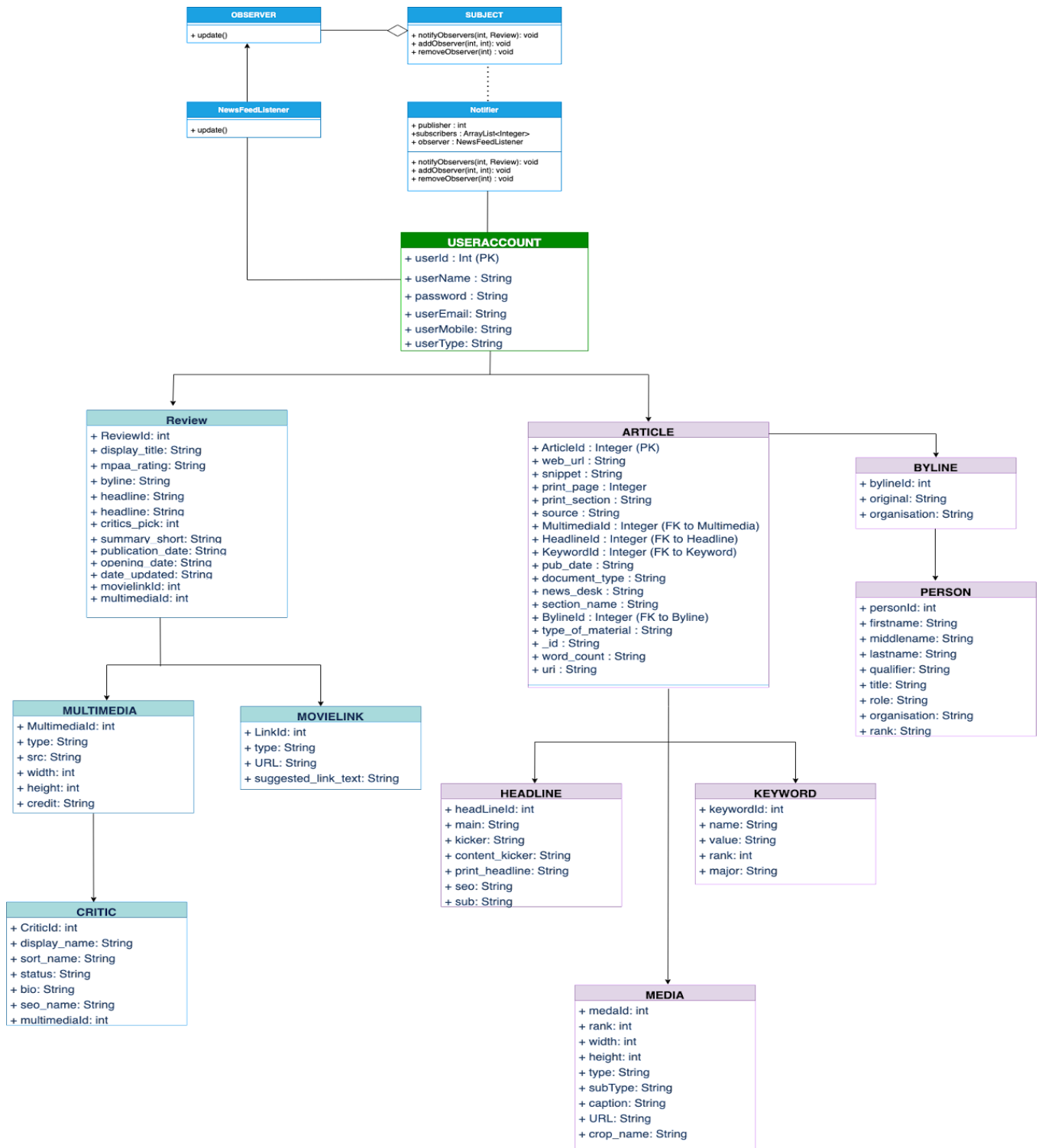


## 2.1 LOGICAL VIEW

---

This section describes the architecturally significant parts of the design model, such as its decomposition into classes and packages and the interfaces and for each significant package, its decomposition into classes and class utilities. It also explained about the class attributes and the objects associated with the class. The interface and observer part and displaying the observer design patter.

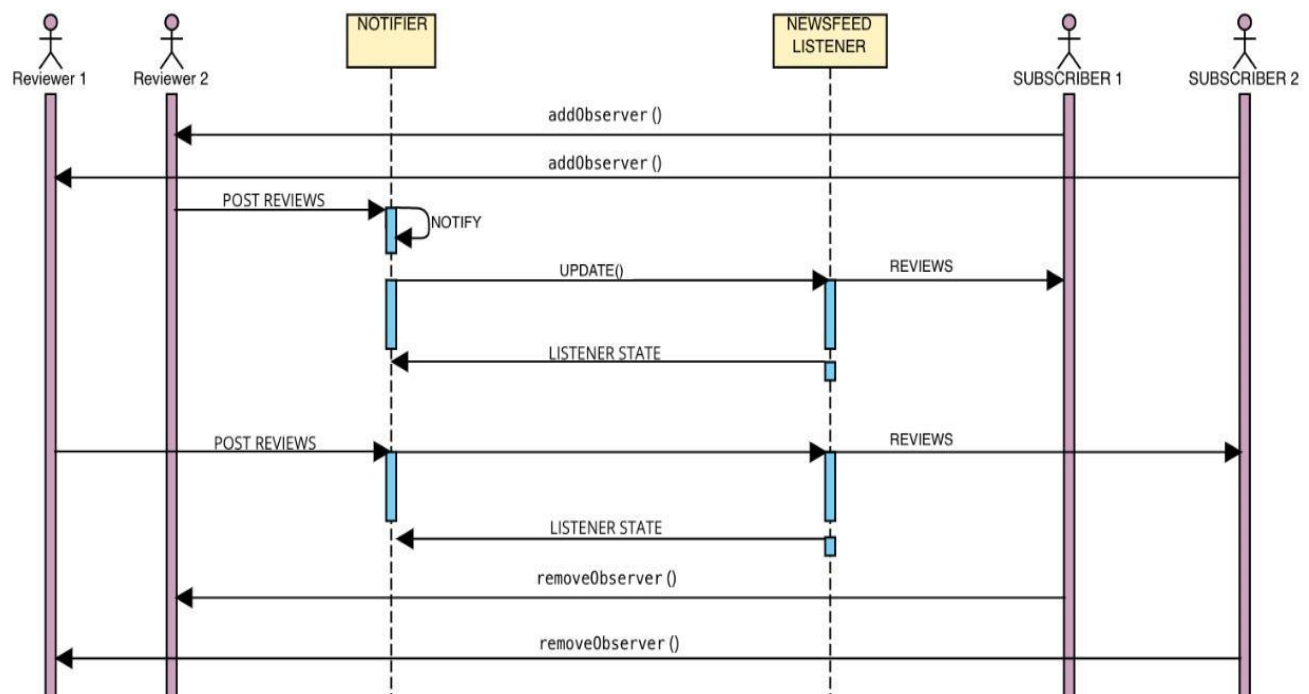
The below given diagram is explaining the 2 API's data fetched from internet i.e, Movie Reviews and Articles. We have implemented review table and all the table data associated with it in our project.





## 2.2 PROCESS VIEW

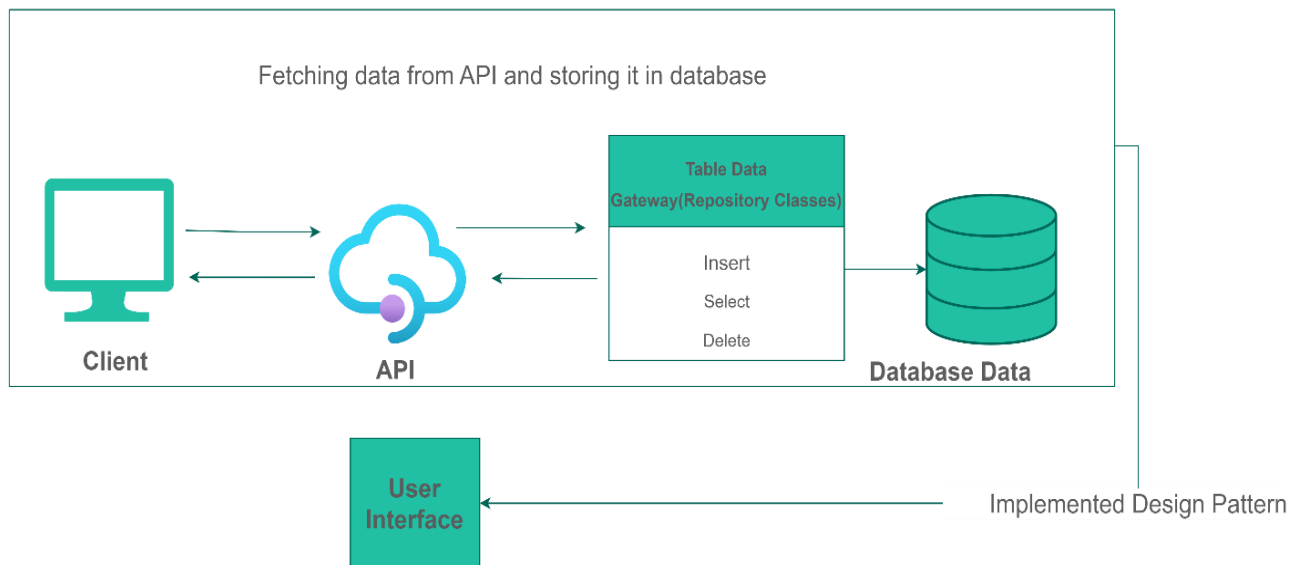
This describes the ongoing picture inside the application from programmer's perspective. In the below given diagram, two reviewers has been defined along with two subscribers. The subscribers are allowed to subscribe to the reviewers they wish to add. The NewsFeedListener are used to communicate with the corresponding reviewer. Once the Subscriber has subscribed to them, whenever the Corresponding reviewer has posted any content, the subscriber will receive a notification regarding it and can view the data.



## 2.3 DEPLOYMENT VIEW

---

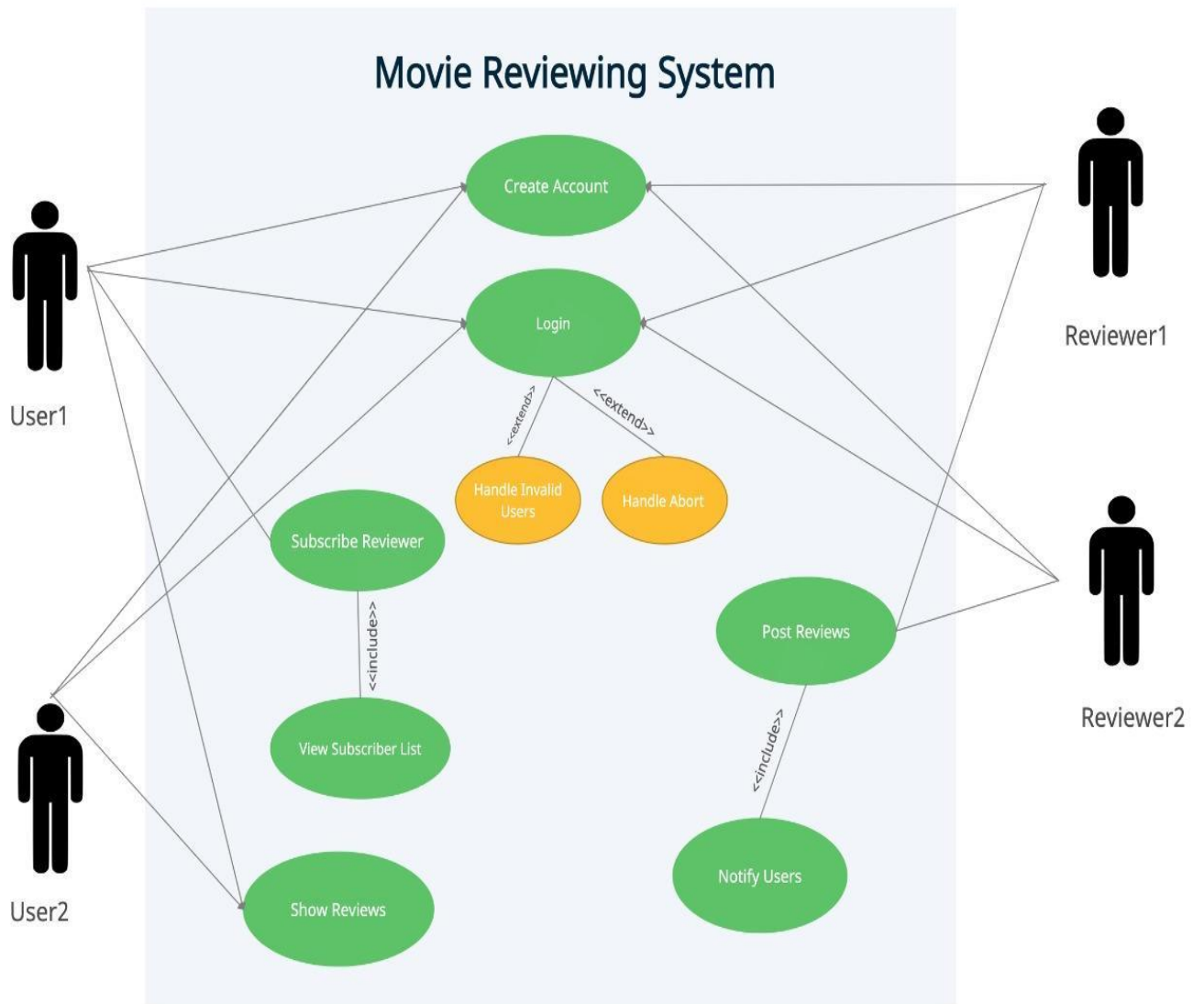
This describes the high-level picture of the project in which client wants to review and post the reviews about the movies available in the API data and then he wanted to store that in database so that their users can read about them.



## 2.4 USE CASE DIAGRAM

---

A **use case diagram** is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures. A user can either signup as a reviewer or a user and then accordingly they can login with their credentials which are stored in UserAccount table. Various operations are shown in the diagram given below.

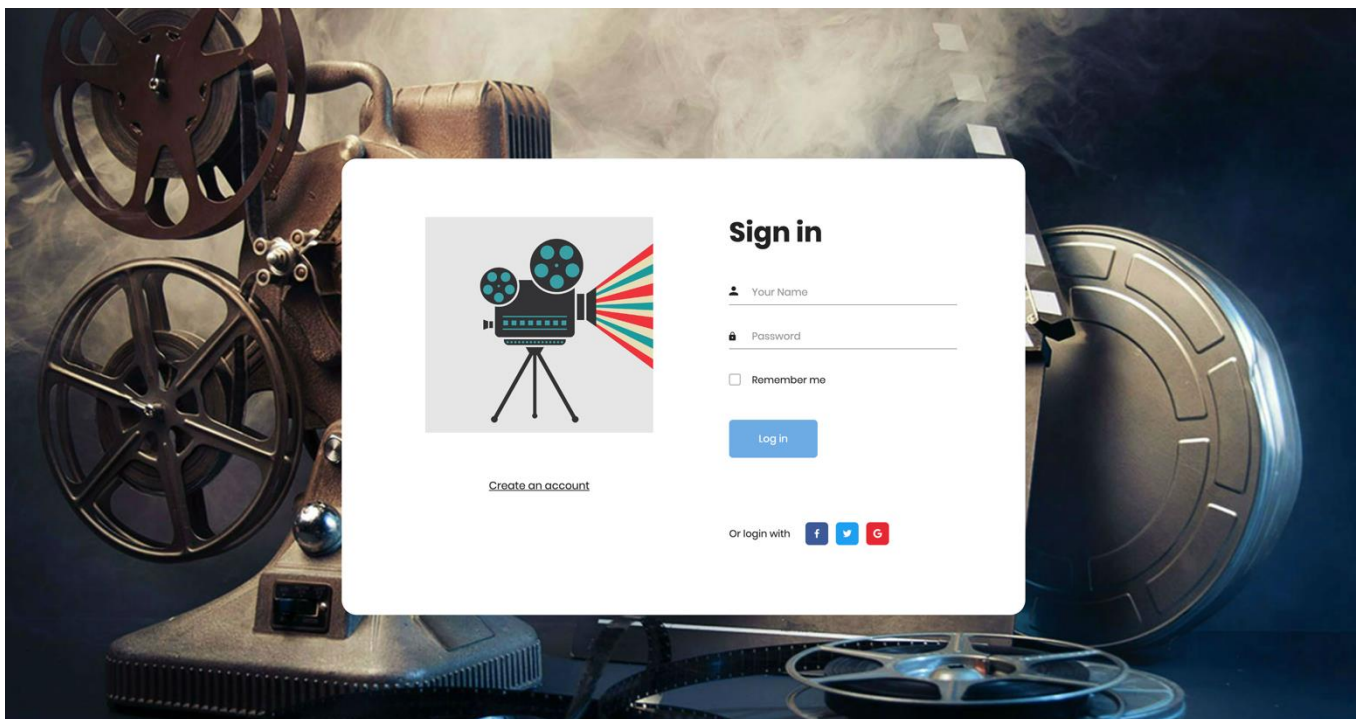


## 3. USER INTERFACE DESIGN

---

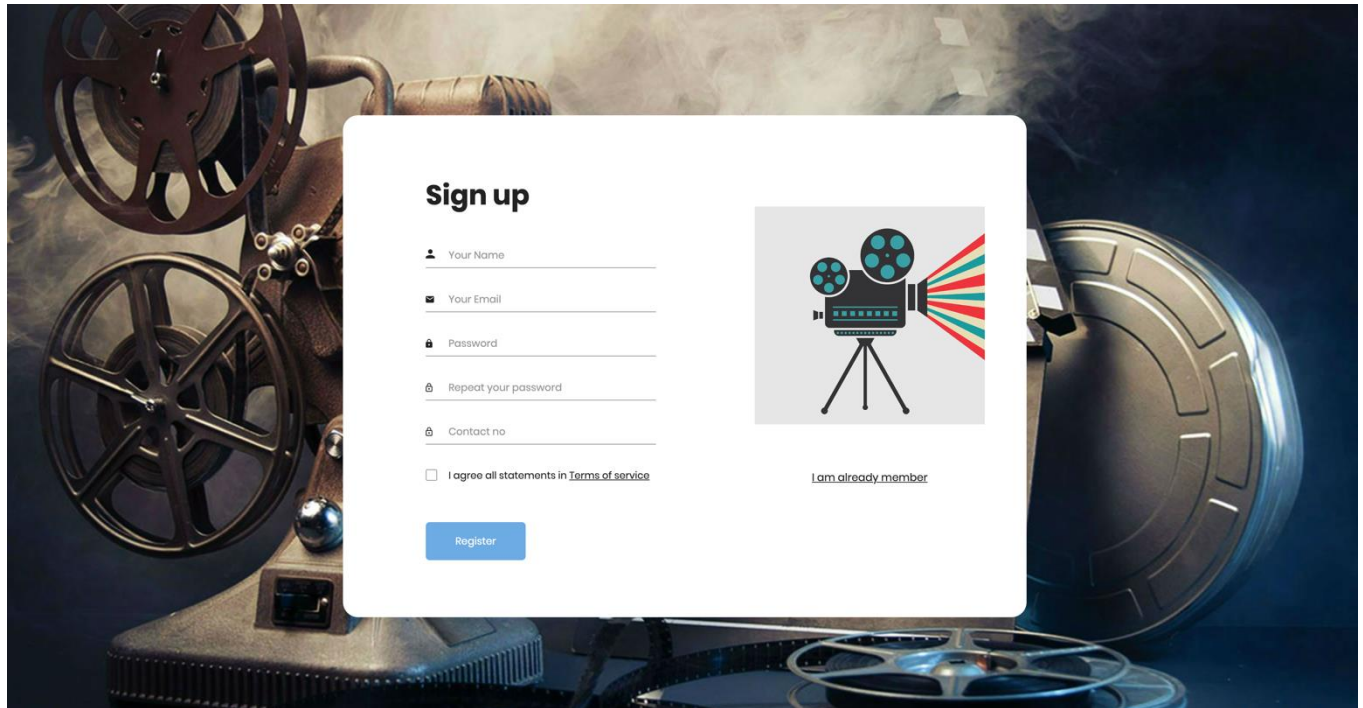
### 3.1 LOGIN PAGE

---



The Login page is used for both Reviewer and Subscriber. The internal view of both user is differentiated when it is navigated inside. The sign in through facebook, twitter and google are not yet added and can be implemented in the later stages.

## 3.2 REGISTRATION PAGE



**Sign up**

Your Name

Your Email

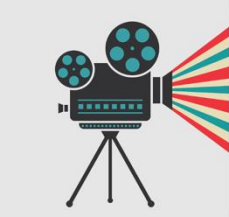
Password

Repeat your password

Contact no

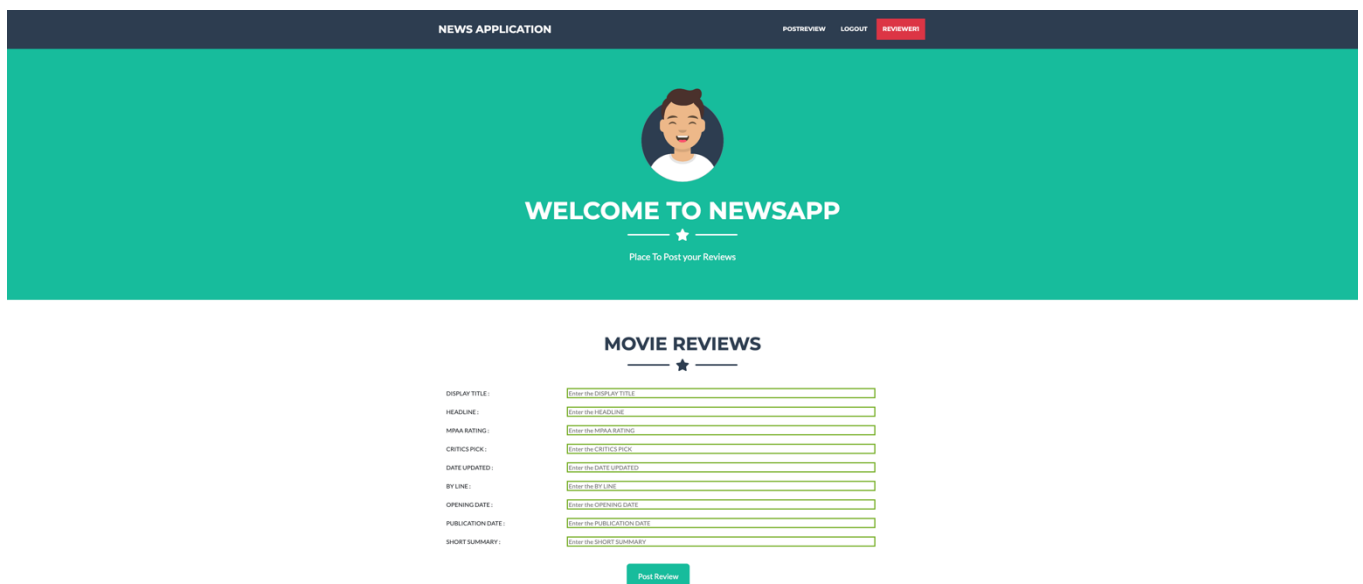
☐ I agree all statements in [Terms of service](#)

[Register](#)




[I am already member](#)

## 3.3 REVIEWER VIEW



NEWS APPLICATION [POST REVIEW](#) [LOGOUT](#) [REVIEWER](#)



**WELCOME TO NEWSAPP**

★

Place To Post your Reviews

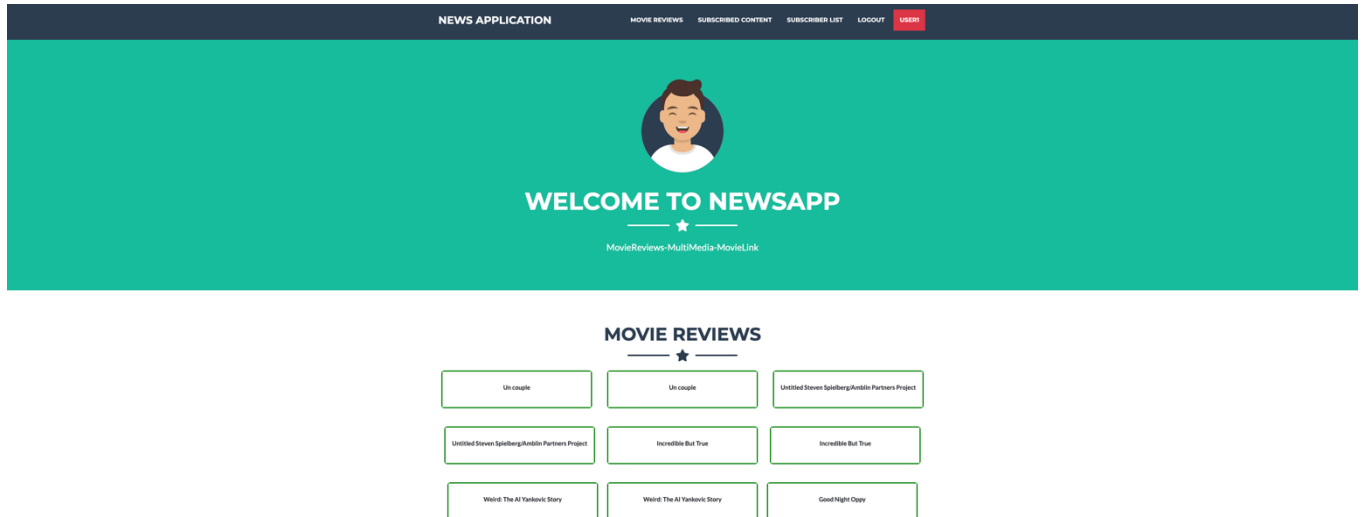
**MOVIE REVIEWS**

★

DISPLAY TITLE:	<input type="text"/>
HEADLINE:	<input type="text"/>
MPIA RATING:	<input type="text"/>
CRITICS PICK:	<input type="text"/>
DATE UPDATED:	<input type="text"/>
BY LINE:	<input type="text"/>
OPENING DATE:	<input type="text"/>
PUBLICATION DATE:	<input type="text"/>
SHORT SUMMARY:	<input type="text"/>

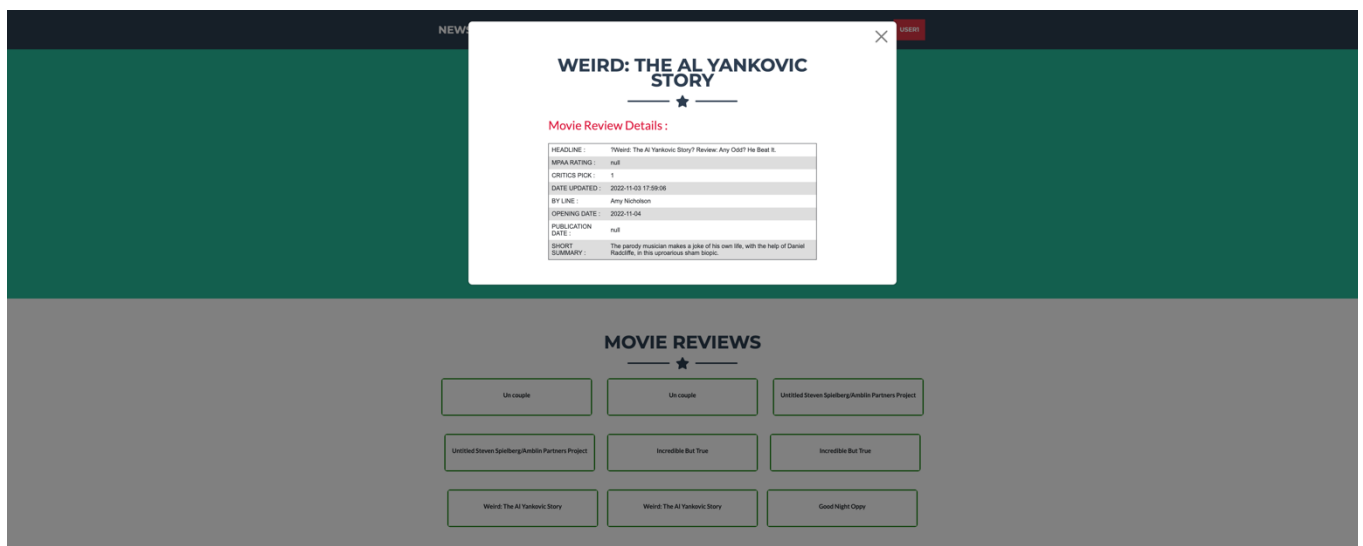
[Post Review](#)

## 3.4 USER VIEW

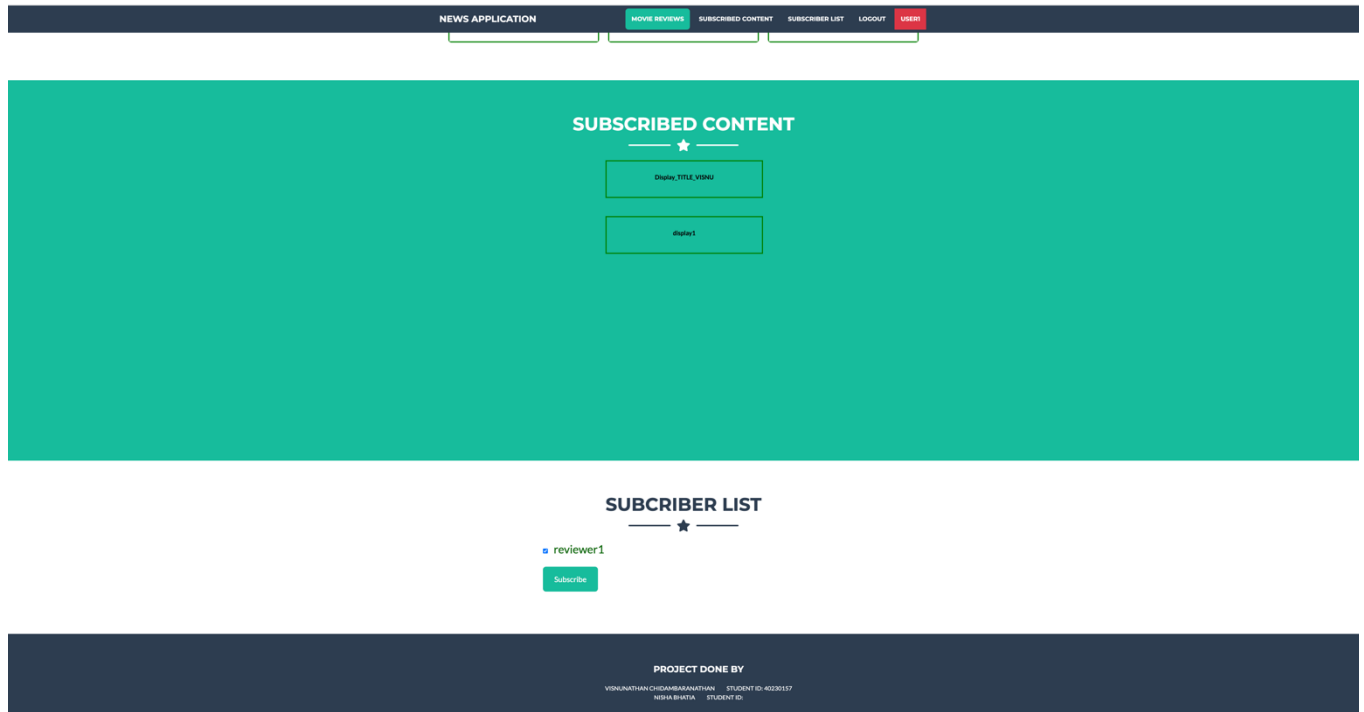


All the movie reviews are displayed (Both the data obtained from API and from the reviewer).

## 3.5 REVIEW DATA LIST



## 3.6 SUBSCRIBING REVIEWER AND SUBSCRIBED DATA LIST



## 4. TESTING

---

To test the functionalities of our system we have used JUNIT and Selenium testing.

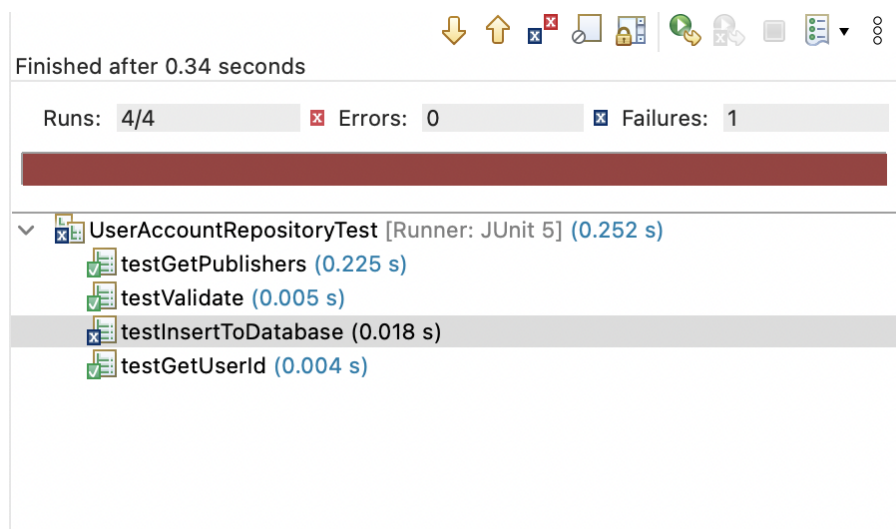
### 4.1 DATABASE

---

Writing a command-line script that runs a MySQL CLI client, and checks the output of the following tests:

- Connection to the database engine succeeds
- Database exists
- All the tables exist
- Definition of each table is as expected

It would also be interesting to test insert, updates, deletes and selects encapsulating them in transactions and rolling back at the end, instead of committing.



#### JUNIT Testing for User Account Repository



## 4.2 WEB SERVER

---

The Java is used to connect with the MySql server. All the connections to the database go through the Table Data Gateway. Junit is used to do the basic unit testing for the connections made.

Having in mind that the web server offers to enter data into the database, some of the tests that could be written are:

- It's possible to log in/out.
- For each entity that is exposed to the RESTful API, it's possible to insert new entries, and only update, select and delete the ones that it has access to.
- Models (internal functions) should be tested too. For example, for the Review model:

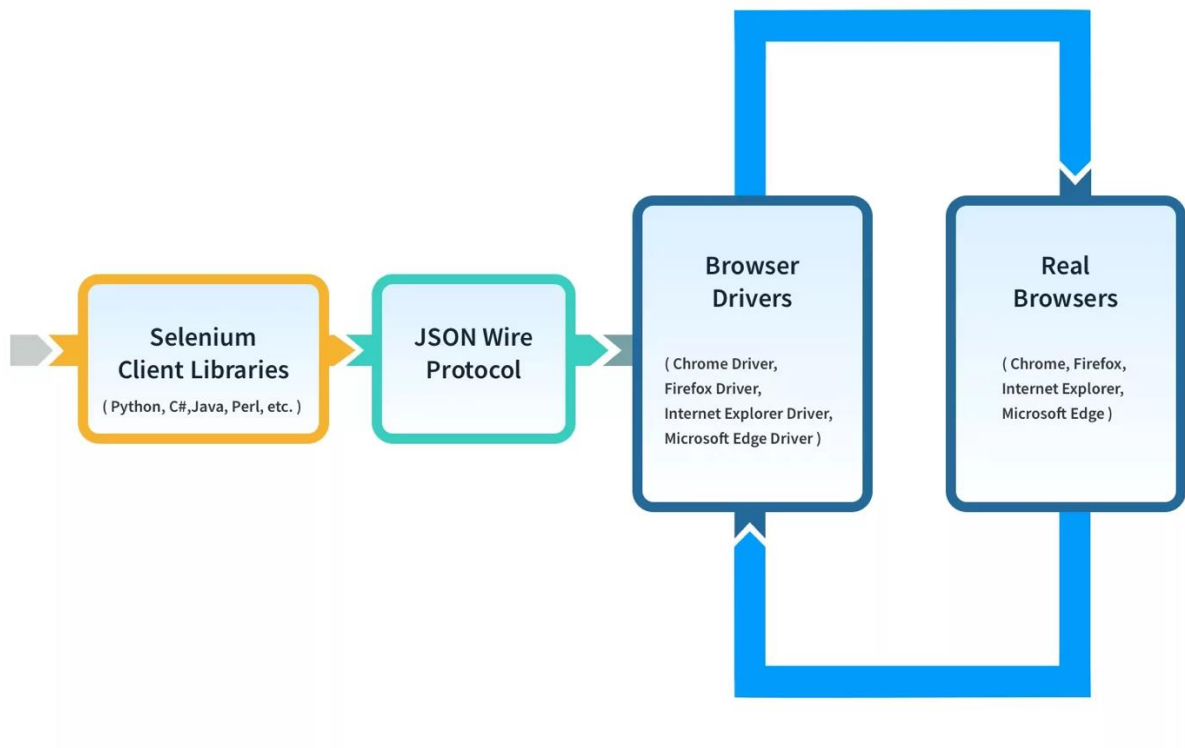
insertToReviewTable is used to insert the values and the testing part will check if the insertion works perfectly fine.

selectAReview is used to retrieve the records from the database and this is tested with the help of Junit by testing the actual value with the expected value.

## 4.3 WEB APPLICATION

---

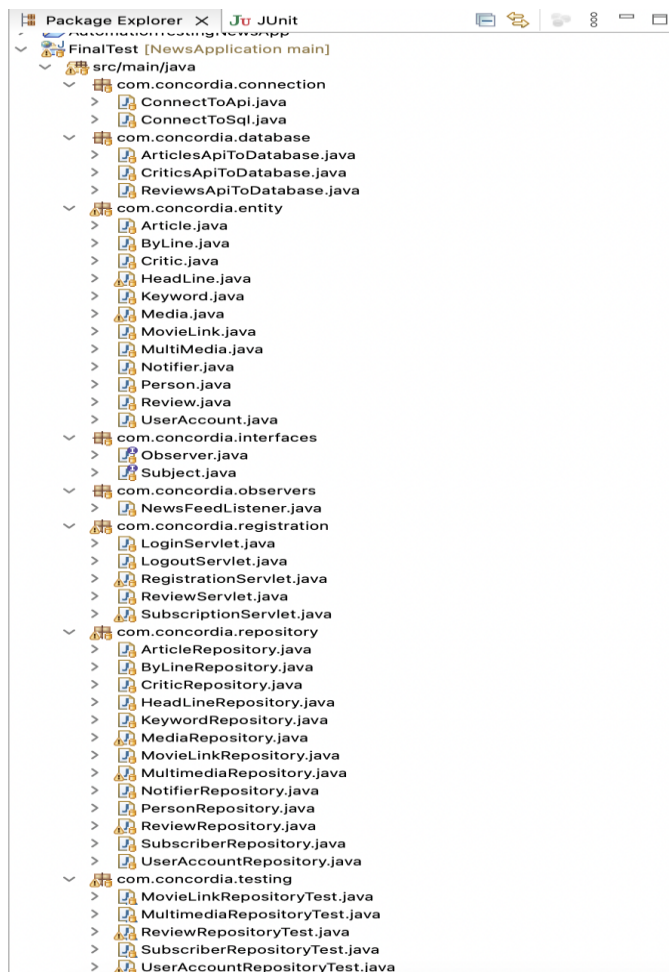
Selenium testing software is a leading automation framework for web applications. There are also Selenium alternatives for testing. Selenium is an umbrella project for a range of tools and libraries that enable and support the automation of web browsers. At the core of Selenium is WebDriver, an interface to write instruction sets that can be run interchangeably in many browsers. We have used selenium testing to automate the test cases through the User Interface. The selenium web driver starts the application by entering all the details as mentioned in the program and checks if any of the cases are failing.



## 5.CODE REFACTORING

---

- All the handling has been done in separate packages and the packages are used to communicate with corresponding works.
- For example: the package `com.concordia.repository` is used as a Data mapper and is used only to handle the queries.
- The package `com.concordia.connection` is the place where all the connection details are established. By this there is no need to change the changes everywhere.



- All the entities are defined under separate packages, and they define the corresponding repository data.
- Public constructor and private constructor are used for object creation.
- Each variable's getters and setters are defined separately to access them.
- By default, the variables are package private and cannot be accessed apart from getters and setters.
- Separate function has been defined to set the JSON object to this class.
- Proper comments have been defined for the functions.

```

package com.concordia.entity;

import org.json.simple.JSONObject;

public class MovieLink {
    int Linkid;
    String Type;
    String URL;
    String suggested_link_text;

    public MovieLink() {
    }

    public MovieLink(String type, String uRL, String suggested_link_text) {
        Type = type;
        URL = uRL;
        this.suggested_link_text = suggested_link_text;
    }

    public int getLinkid() {
        return Linkid;
    }

    public void setLinkid(int linkid) {
        Linkid = linkid;
    }

    public String getType() {
        return Type;
    }

    public void setType(String type) {
        Type = type;
    }

    public String getURL() {
        return URL;
    }

    public void setURL(String uRL) {
        URL = uRL;
    }

    public String getSuggested_link_text() {
        return suggested_link_text;
    }

    public void setSuggested_link_text(String suggested_link_text) {
        this.suggested_link_text = suggested_link_text;
    }

    /**
     * @param linkData
     * @return MovieLink object with all the data in it
     */
    public static MovieLink setAllMovieLinkData(JSONObject linkData) {
        MovieLink movielink = new MovieLink();

        if(linkData != null) {
            if(linkData.get("type")!=null && !(linkData.get("type").toString().isEmpty())) {
                movielink.setType(linkData.get("type").toString());
            }

            if(linkData.get("url")!=null && !(linkData.get("url").toString().isEmpty())) {
                movielink.setURL(linkData.get("url").toString());
            }

            if(linkData.get("suggested_link_text")!=null && !(linkData.get("suggested_link_text").toString().isEmpty())) {
                movielink.setSuggested_link_text(linkData.get("suggested_link_text").toString());
            }
        }

        return movielink;
    }
}

```

```

package com.concordia.connection;

import java.sql.Connection;

public class ConnectToSql {

    private static String dbUrl = "jdbc:mysql://localhost:3306/NewsAppDB?allowPublicKeyRetrieval=true&useSSL=false&useUnicode=true&characterEncoding=utf8";
    private static String dbName = "root";
    private static String dbPassword = "root@123";
    private static String dbDriver = "com.mysql.cj.jdbc.Driver";

    public static void loadDriver()
    {
        try {
            Class.forName(dbDriver);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public static Connection getConnection()
    {
        Connection con = null;
        try {
            con = DriverManager.getConnection(dbUrl, dbName, dbPassword);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return con;
    }
}

```

- Extracting proper interface from the code, so that it can be used to achieve abstraction. It is mainly used for **Capturing similarities among unrelated classes without artificially forcing a class relationship.**

```

SubscriberRepos  registration.js  LoginServlet.java  UserAccountRepo  Media.java  MovieLink.java  Observer.java X 41
1 package com.concordia.interfaces;
2
3 import com.concordia.entity.Review;
4
5 public interface Observer {
6
7     public void update(Integer publisherId, Review review);
8     public void addSubscriberToPublisher(Integer publisherId, Integer subscriberId);
9     public void removeSubscriberToPublisher(Integer subscriberId);
10
11 }
12

```

- Testing is done under separate package to remove the dependency.
- Static import is used which allows to access the static members of a class without the class qualification.

- ▼ com.concordia.testing
  - > MovieLinkRepositoryTest.java
  - > MultimediaRepositoryTest.java
  - > ReviewRepositoryTest.java
  - > SubscriberRepositoryTest.java
  - > UserAccountRepositoryTest.java

```
package com.concordia.testing;

import static org.junit.jupiter.api.Assertions.*;

class MovieLinkRepositoryTest {

    MovieLink movielink = new MovieLink("article1", "https://www.nytimes.com/2022/11/03/movies/weird-the-al-yankovic-story-review.h

    @Test
    void testInsertToMovieLinkTable() {
        Integer value = MovieLinkRepository.insertToMovieLinkTable(movielink);
        Assert.assertTrue(value instanceof Integer);
    }

    @Test
    void testSelectFromMovieLink() {
        MovieLink actual = MovieLinkRepository.selectFromMovieLink(1645);
        assertEquals(actual.getURL(), movielink.getURL());
    }
}
```

- The methods are properly overloaded when the class is implementing a parent class.

```
package com.concordia.observers;

import com.concordia.entity.Review;

public class NewsFeedListener implements Observer{

    @Override
    public void update(Integer publisherId, Review review) {
        try {
            Integer reviewId = ReviewRepository.insertToReviewTable(review);
            SubscriberRepository.insertSubscriberData(publisherId, reviewId);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void addSubscriberToPublisher(Integer subscriberId, Integer publisherId) {
        try {
            NotifierRepository.insertSubscriberData(subscriberId, publisherId);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void removeSubscriberToPublisher(Integer subscriberId) {
        try {
            NotifierRepository.deletePublisherData(subscriberId);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

1.

```

1 package com.concordia.repository;
2
3 import java.sql.Connection;
4
11 public class UserAccountRepository {
12
13     /**
14      *
15      * @param useraccount
16      * @return userId of type String
17      */
18     public String getUserId(UserAccount useraccount)
19     {
20         String id = "";
21
22         ConnectToSql.loadDriver();
23         Connection con = ConnectToSql.getConnection();
24
25         String sql = "select userid from UserAccount where username = ? and password = ?";
26         PreparedStatement ps;
27         try {
28             ps = con.prepareStatement(sql);
29             ps.setString(1, useraccount.getUsername());
30             ps.setString(2, useraccount.getPassword());
31             ResultSet rs = ps.executeQuery();
32             while(rs.next()) {
33                 id = rs.getString("userid");
34             }
35         } catch (SQLException e) {
36             e.printStackTrace();
37         } finally {
38             if (con != null)
39                 try {
40                     con.close();
41                 } catch (SQLException ignore) {
42                     ignore.printStackTrace();
43                 }
44         }
45         return id;
46     }
47
48     /**
49      *
50      * @param useraccount
51      * @return boolean value which mentions the success or failure
52      */
53     public boolean validate(UserAccount useraccount)
54     {
55         boolean status = false;
56
57         ConnectToSql.loadDriver();
58         Connection con = ConnectToSql.getConnection();
59
60         String sql = "select * from UserAccount where username = ? and password = ?";
61         PreparedStatement ps;
62         try {
63             ps = con.prepareStatement(sql);
64             ps.setString(1, useraccount.getUsername());
65             ps.setString(2, useraccount.getPassword());
66             ResultSet rs = ps.executeQuery();
67             status = rs.next();
68         } catch (SQLException e) {
69             e.printStackTrace();
70         }
71     }
72 }

```

- Proper comments are maintained throughout the code which gives an overview of the method, parameter list and the corresponding return value.

## 6.GITHUB REPOSITORY

<https://github.com/rocketvisnu/NewsApplication>

The screenshot shows the GitHub repository page for `rocketvisnu/NewsApplication`. The repository is private and has 0 stars, 1 watcher, and 0 forks. The main branch is `main` with 2 branches and 0 tags. The repository contains 41 commits. The file list shows the following files and their commit history:

File	Commit	Time
.settings	UI changes	4 days ago
AutomationTestingNewsApp	Code refactoring	3 days ago
FinalTest	Added Repo Files	4 days ago
src	Code Refactoring	5 hours ago
.DS_Store	Code Refactoring	5 hours ago
.classpath	UI changes	4 days ago
.gitignore	UI changes	4 days ago
.project	Added Repo Files	4 days ago
SRS_Document.docx	UI changes	4 days ago
architecture diagram.png	Architectural Diagrams	3 days ago
implementation diagram.png	Architectural Diagrams	3 days ago
use case diagram.jpg	Architectural Diagrams	3 days ago

The repository also includes a README section with a button to [Add a README](#). The right sidebar shows the repository's metadata, including the `About` section (No description or website provided), `Releases` (No releases published), `Packages` (No packages published), `Contributors` (2 contributors: nishabhatia98 and rocketvisnu Visnunathan), and `Languages` (CSS 60.4%, Java 25.4%, HTML 7.9%, SCSS 4.6%, JavaScript 1.7%).



## 7. REFERENCE

---

- <https://docs.oracle.com/javaee/5/tutorial/doc/bnagy.html>
- <https://docs.oracle.com/javaee/5/tutorial/doc/bnafe.html>
- <https://tomcat.apache.org/>
- <https://www.mysql.com/>
- <https://html.com/>
- <https://developer.mozilla.org/en-US/docs/Web/CSS>
- <https://www.javascript.com/>
- <https://learn.microsoft.com/en-us/dotnet/standard/events/observer-design-pattern>
- <https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html>
- <https://junit.org/junit5/>
- <https://www.java.com/en/>
- <https://www.selenium.dev/documentation/>
- <https://drive.google.com/file/d/19M2bD5uUHralrzPex14ilyG0N1sYwN8T/view>