IT 303 – Software Verification, Validation and Testing

TESTING DOCUMENTATION

IMDB Test

Prepared by:
Andrej Herman

Proposed to:
Samed Jukić, Assist. Prof. Dr.
Adnan Miljković, Teaching Assistant
Benjamin Peljto, Lecture Assistant

9th January, 2025

TABLE OF CONTENTS

1. Introduction

1.1. About the Project

The webpage I am testing is IMDb. According to the Wikipedia, it's an online database of information related to films, television series, podcasts, home videos, video games, and streaming content online – including cast, production crew and personal biographies, plot summaries, trivia, ratings, and fan and critical reviews.

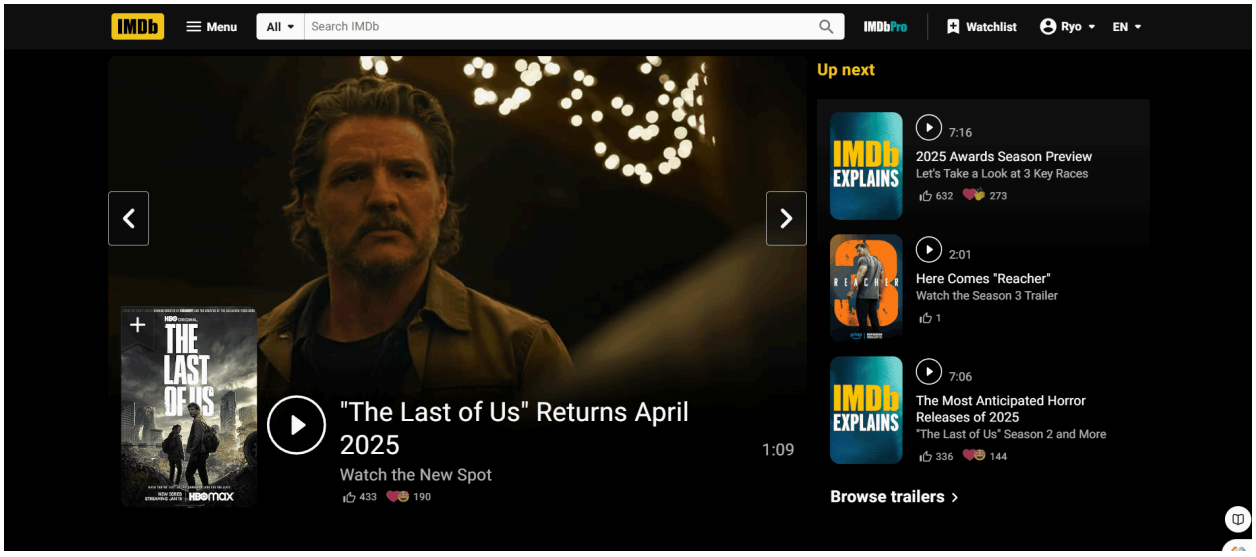
1.2. Project Functionalities and Screenshots

Some of the main IMDb functionalities are:

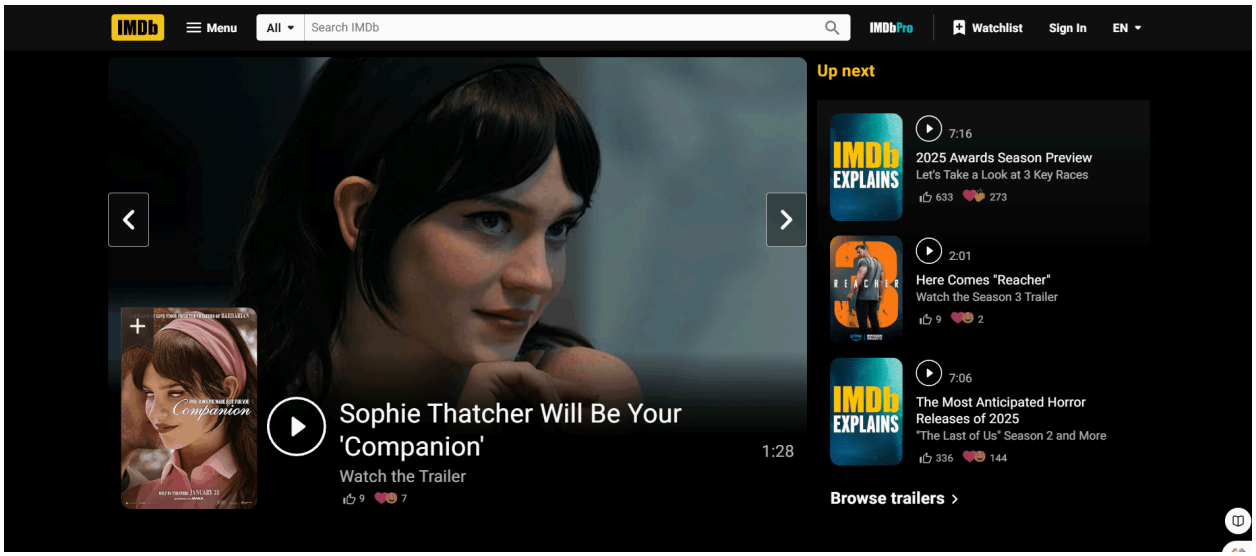
1. **Search Functionality**: Users can search for movies, TV shows, celebrities, etc.
2. **User Authentication**: Users can register, log in, and manage their profiles.
3. **Title Ratings and Reviews**: Users can rate and review titles.
4. **Watchlist**: Users can add movies to their watchlist.
5. **Localization**: Users can change a language to their preference.
6. **Navigation**: Various categories and filters to browse content.
7. **Trailers and Media**: Access to trailers, images, and other media for movies and TV shows.

This is how some of them look like:

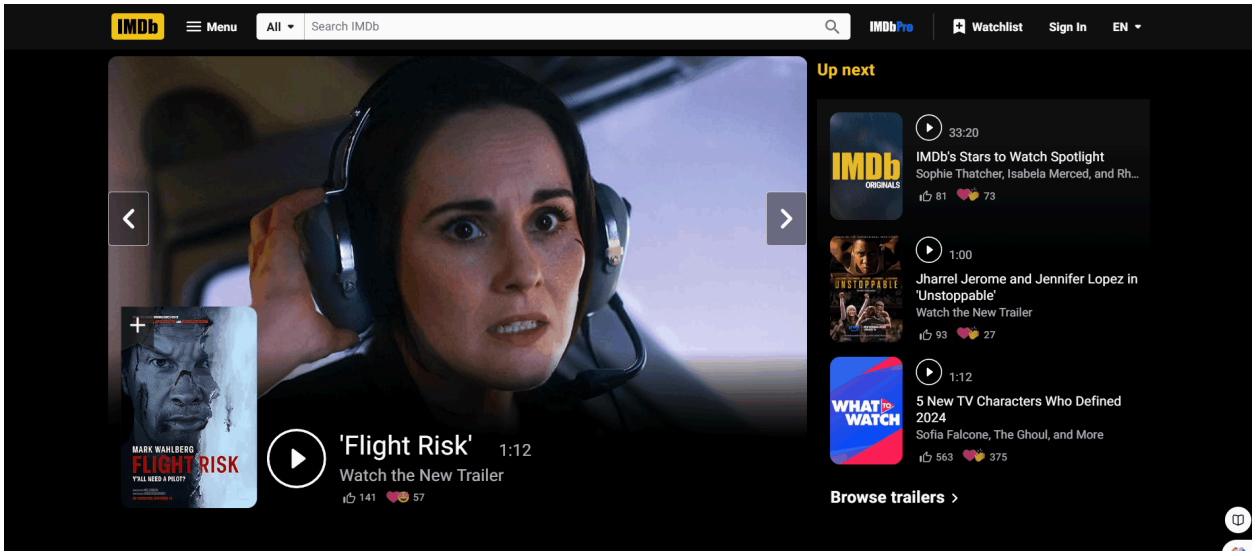
Search functionality



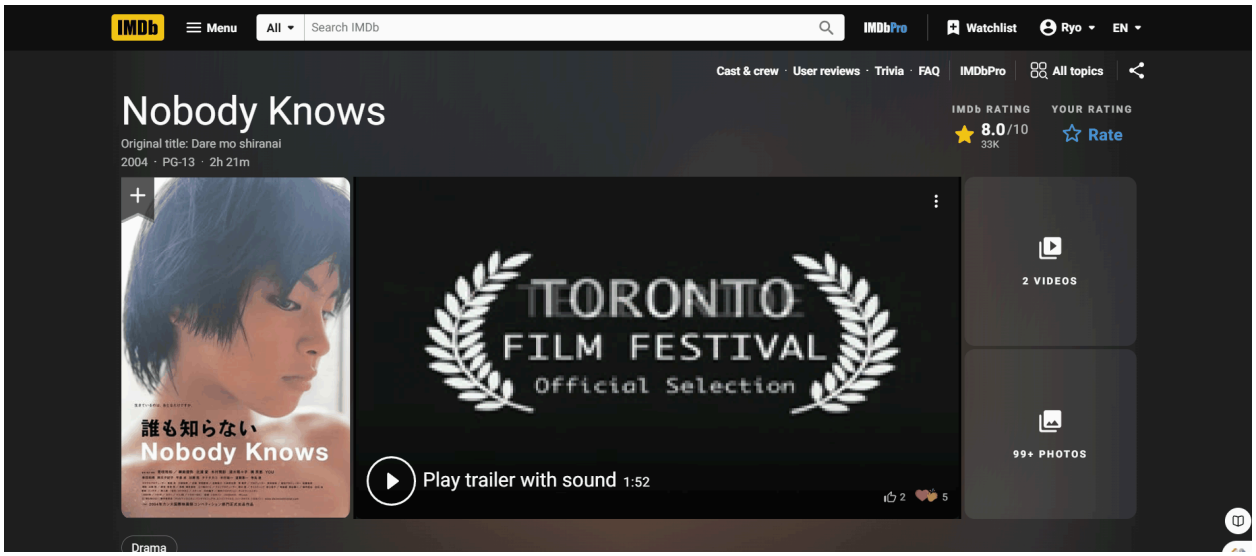
User authentication



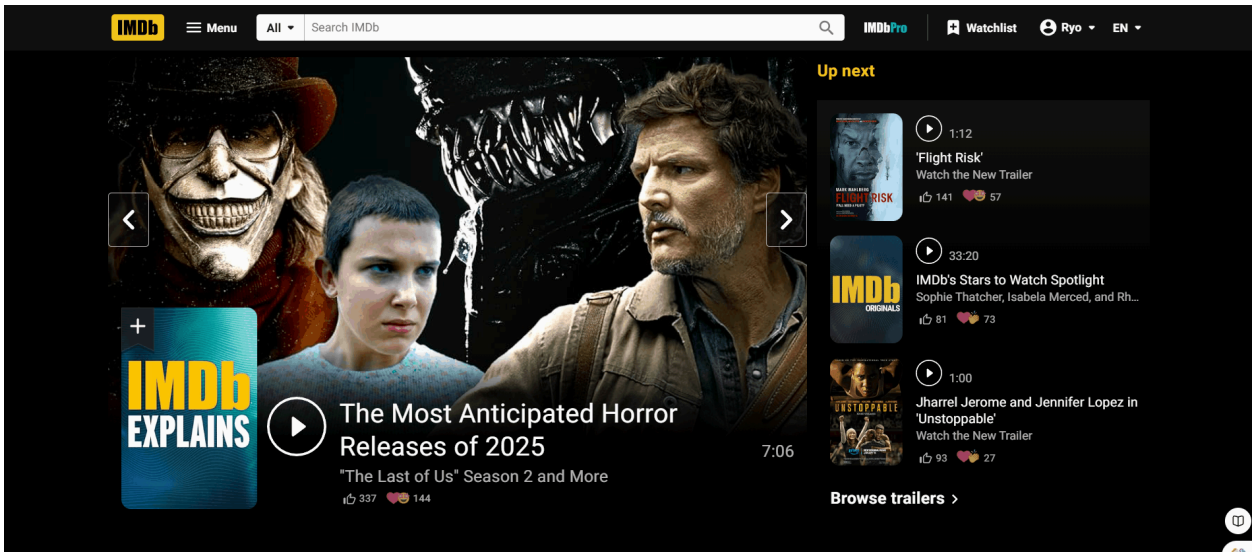
Navigation



Watchlist and Title Ratings



Localization



2. Test Plan

2.1. Scope

The scope of this automated test implementation includes essential testing of UI elements, UX, security and performance. I have tried to cover testing of all of the crucial webpage functionalities. I couldn't cover testing all of the UI elements because there are too many and that amount is out of the scope for this project.

2.2. Testing Environment and Tools

Tools, frameworks and libraries: Selenium WebDriver (ChromeDriver), WebDriverManager library, JUnit, IntelliJ IDEA.

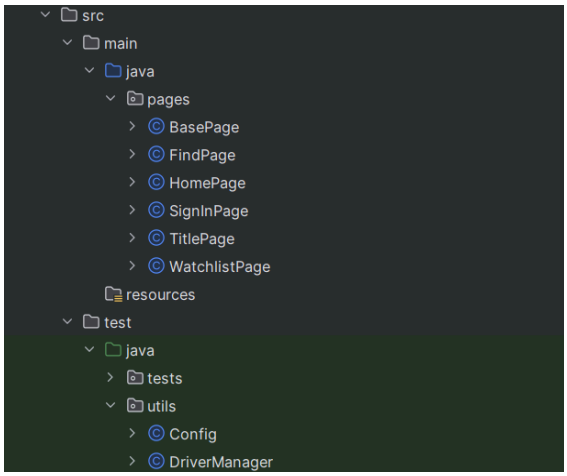
Programming language: Java.

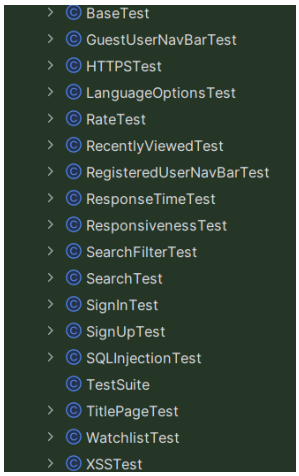
Testing environment: Google Chrome on Windows 11.

2.3. Project Design

The project is designed to follow the best development practices. I have implemented Page Object Model design pattern which models web elements and all the functions needed in test cases in separate page classes improving readability, maintenance, reusability and performance. It is important to mention that I didn't fully implement POM, I have just taken some principles from it, like making page models, making base classes and maintaining separation of concerns. I have implemented separate Config class which contains crucial constant data - base URL, email address and password. I have also added the DriverManager class which handles WebDriver setup, initialization and deinitialization. I have preserved test independency, browser restart for every test and implemented cookie insertion mechanism to avoid continuous signing in avoiding bot detection.

2.3.1 File structure





2.3.2 Page models

For every page accessed on IMDb I have made a page model class which contains selectors for web elements and functions which are not atomic. Every page class inherits BasePage class which contains reusable code like PageFactory which initializes @FindBy annotated web elements, WebDriver and certain often-used functions.

BasePage class

```
public class BasePage {
    protected WebDriver driver;
    protected WebDriverWait wait;

    public BasePage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }

    protected void scrollToElement(WebElement element) {
                                        ((org.openqa.selenium.JavascriptExecutor)
driver).executeScript("arguments[0].scrollIntoView(true);", element);
    }
```

```
    protected void implicitWait(int ms) {
        try {
            Thread.sleep(ms);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            System.err.println("Sleep interrupted: " + e.getMessage());
        }
    }
}
```

HomePage class

```
public class HomePage extends BasePage {
    public HomePage(WebDriver driver) {
        super(driver);
    }

    @FindBy(id = "home_img_holder")
    private WebElement navBarLogo;

    @FindBy(id = "imdbHeader-navDrawerOpen")
    private WebElement navBarMenu;

    @FindBy(css = ".ipc-btn--core-base")
    private WebElement navBarSearchFilter;

    @FindBy(css = "#nav-search-form > div:nth-child(1) > div > div > div > div >
ul > li:nth-child(2)")
    private WebElement navBarSearchTitleFilter;

    @FindBy(css = ".gUPgwv.drawer")
    private WebElement navBarMenuShown;

    @FindBy(id = "suggestion-search")
    private WebElement navBarSearch;

    @FindBy(css = "#suggestion-search-button")
```

```
    private WebElement navBarSearchFindButton;

    @FindBy(css = "div.navbar__imdbpro.sc-GvgMv.ftVCet > div > a")
    private WebElement navBarIMDBProIcon;

    @FindBy(css = "div.sc-fcmLWC.jzOrdI.imdb-header__watchlist-button > a")
    private WebElement navBarWatchlistButton;

    @FindBy(css = "div.nav__userMenu.navbar__user.sc-fuISYh.hrzrxR > a")
    private WebElement navBarSignInButton;

                                                    @FindBy(css                    =
"div.sc-jQAyio.sc-csTbTw.dUVOWS.cXNOwG.navbar__flyout--breakpoint-m > label")
    private WebElement navBarLanguageSelectorButton;

    @FindBy(id = "language-option-en-US")
    private WebElement languageSelectorEnglish;

    @FindBy(id = "language-option-de-DE")
    private WebElement languageSelectorGerman;

    @FindBy(id = "language-option-it-IT")
    private WebElement languageSelectorItalian;

    @FindBy(id = "nav-language-selector-contents")
    private WebElement navBarLanguageMenu;

    // Shown only if user is logged in
    @FindBy(css = ".navbar__user-menu-toggle__button")
    private WebElement navBarProfileIcon;

    // Shown only if user is logged in
    @FindBy(id = "navUserMenu-contents")
    private WebElement navBarProfileMenu;

    // Shown only if user is logged in
    @FindBy(css = "#navUserMenu-contents > ul > a:nth-child(3)")
    private WebElement navBarProfileMenuProfile;
```

```
                                                    @FindBy(css =
"div.ipc-shoveler.ipc-shoveler--base.ipc-shoveler--page0.rvi-shoveler")
    private WebElement recentlyViewedSection;

                                                    @FindBy(css =
"div.ipc-title.ipc-title--baseAlt.ipc-title--subsection-title.ipc-title--on-tex
tPrimary.sc-10b3b195-0.cxKMzQ > div > div > button")
    private WebElement clearRecentlyViewed;

    @FindBy(id = "captcha-container")
    private WebElement captchaContainer;

    public WebElement getCaptchaContainer() {
        return captchaContainer;
    }

    public WebElement getNavBarLogo() {
        return navBarLogo;
    }

    public WebElement getClearRecentlyViewed() {
        return clearRecentlyViewed;
    }

    public WebElement getNavBarSearch() {
        return navBarSearch;
    }

    public WebElement getNavBarIMDBProIcon() {
        return navBarIMDBProIcon;
    }

    public WebElement getNavBarWatchlistButton() {
        return navBarWatchlistButton;
    }

    public WebElement getNavBarSignInButton() {
```

```
        return navBarSignInButton;
    }

    public WebElement getNavBarMenu() {
        return navBarMenu;
    }

    public WebElement getNavBarMenuShown() {
        return navBarMenuShown;
    }

    public WebElement getNavBarLanguageSelectorButton() {
        return navBarLanguageSelectorButton;
    }

    public WebElement getNavBarLanguageMenu() {
        return navBarLanguageMenu;
    }

    public WebElement getNavBarProfileIcon() {
        return navBarProfileIcon;
    }

    public WebElement getNavBarProfileMenu() {
        return navBarProfileMenu;
    }

    public WebElement getNavBarProfileMenuProfile() {
        return navBarProfileMenuProfile;
    }

    public WebElement getRecentlyViewedSection() {
        return recentlyViewedSection;
    }

    public void searchForText(String query) {
        navBarSearch.sendKeys(query);
        navBarSearchFindButton.click();
```

```
    }

    public void searchForTextTitleFilter(String query) {
        navBarSearch.sendKeys(query);
        navBarSearchFilter.click();
        navBarSearchTitleFilter.click();
        navBarSearchFindButton.click();
    }

    public void goToRecentlyViewedSection() {
        scrollToElement(recentlyViewedSection);

        implicitWait(500);
    }

    public void switchLanguage(String language) {
        navBarLanguageSelectorButton.click();
        JavascriptExecutor js = (JavascriptExecutor) driver;

        switch(language) {
            case "English":
                                        js.executeScript("arguments[0].click();",
languageSelectorEnglish);
                break;
            case "German":
                                        js.executeScript("arguments[0].click();",
languageSelectorGerman);
                break;
            case "Italian":
                                        js.executeScript("arguments[0].click();",
languageSelectorItalian);
                break;
            default:
                        System.out.println("Choose between English, German and
Italian!");
        }
    }
}
```

SignInPage class

```
public class SignInPage extends BasePage {
    public SignInPage(WebDriver driver) {
        super(driver);
    }

    @FindBy(css = "#signin-options > div > div:nth-child(2) > a:nth-child(1)")
    private WebElement signInWithImdb;

    @FindBy(id = "ap_email")
    private WebElement emailField;

    @FindBy(id = "ap_password")
    private WebElement passwordField;

    @FindBy(id = "signInSubmit")
    private WebElement signInSubmit;

    @FindBy(id = "auth-error-message-box")
    private WebElement authErrorMsgBox;

    @FindBy(css = "#auth-show-password-checkbox")
    private WebElement showPasswordBox;

    @FindBy(css = "#auth-password-container > div > span")
    private WebElement visiblePasswordContainer;

    @FindBy(css = "#signin-options > div > div:nth-child(4) > a")
    private WebElement createNewAccountButton;

    @FindBy(id = "ap_customer_name")
    private WebElement signUpNameField;

    @FindBy(id = "ap_password_check")
    private WebElement reenterPasswordField;
```

```
@FindBy(id = "continue")
private WebElement signUpContinueButton;

@FindBy(id = "auth-email-invalid-email-alert")
private WebElement invalidEmailInput;

@FindBy(id = "auth-password-invalid-password-alert")
private WebElement invalidPasswordInput;

public void signIn(String email, String password) {
    HomePage homePage = new HomePage(driver);

    homePage.getNavBarSignInButton().click();
    signInWithImdb.click();
    emailField.sendKeys(email);
    passwordField.sendKeys(password);
    signInSubmit.click();
}

public void signUp(String name, String email, String password) {
    HomePage homePage = new HomePage(driver);

    homePage.getNavBarSignInButton().click();
    getCreateNewAccountButton().click();
    signUpEnterName(name);
    enterEmail(email);
    enterPassword(password);
    reenterPassword(password);
    getSignUpContinueButton().click();

}

public void signUpEnterName(String name) {
    signUpNameField.sendKeys(name);
}

public void enterEmail(String email) {
```

```
        emailField.sendKeys(email);
    }

    public void enterPassword(String password) {
        passwordField.sendKeys(password);
    }

    public void reenterPassword(String password) {
        reenterPasswordField.sendKeys(password);
    }

    public WebElement getAuthErrorMsgBox() {
        return authErrorMsgBox;
    }

    public WebElement getShowPasswordBox() {
        return showPasswordBox;
    }

    public WebElement getVisiblePasswordContainer() {
        return visiblePasswordContainer;
    }

    public WebElement getPasswordField() {
        return passwordField;
    }

    public WebElement getSignInWithImdb() {
        return signInWithImdb;
    }

    public WebElement getCreateNewAccountButton() {
        return createNewAccountButton;
    }

    public WebElement getSignUpContinueButton() {
        return signUpContinueButton;
    }
```

```
    public WebElement getInvalidEmailInput() {
        return invalidEmailInput;
    }

    public WebElement getInvalidPasswordInput() {
        return invalidPasswordInput;
    }
}
```

FindPage class

```
public class FindPage extends BasePage {
    public FindPage(WebDriver driver) {
        super(driver);
    }

    @FindBy(css = "[data-testid='find-results-section-name']")
    private WebElement actorSection;

    @FindBy(css = "[data-testid='find-results-section-title']")
    private WebElement titleSection;

    @FindBy(css = "[data-testid='find-results-section-interest']")
    private WebElement interestSection;

    @FindBy(css = "[data-testid='find-results-section-name']")
    private WebElement peopleSection;

    @FindBy(css = "div.sc-b03627f1-2.gWHDBT > ul > li:nth-child(1)")
    private WebElement titleSectionFirstMatch;

    public WebElement getInterestSection() {
        return interestSection;
    }

    public WebElement getActorSection() {
```

```
        return actorSection;
    }

    public WebElement getTitleSection() {
        return titleSection;
    }

    public WebElement getPeopleSection() {
        return peopleSection;
    }

    public void FindTitle(String titleName) {
        HomePage homePage = new HomePage(driver);
        homePage.searchForText(titleName);
        titleSectionFirstMatch.click();
    }
}
```

TitlePage class

```
public class TitlePage extends BasePage {
    public TitlePage(WebDriver driver) {
        super(driver);
    }

    @FindBy(css = "[data-testid='hero__primary-text']")
    private WebElement titleName;

    @FindBy(css = "div.sc-70a366cc-0.bxYZmb > div")
    private WebElement originalTitleName;

    @FindBy(css = "div.sc-70a366cc-0.bxYZmb > ul")
    private WebElement titleDetails;

    // https://www.imdb.com/title/tt0408664/ratings/?ref_=tt_ov_rat
        @FindBy(css = "div.sc-3a4309f8-0.jJkxPn.sc-70a366cc-1.kUfGfN > div >
div:nth-child(1) > a")
```

```
    private WebElement titleRating;

    @FindBy(css = "[data-testid='hero-rating-bar__user-rating__score']")
    private WebElement titleUserRating;

        @FindBy(css  =  "div.sc-3a4309f8-0.jJkxPn.sc-70a366cc-1.kUfGfN  >  div  >
div:nth-child(2) > button")
    private WebElement rateButton;

    @FindBy(css = "[data-testid='promptable__pc']")
    private WebElement ratePromptWindow;

                                                    @FindBy(css                =
"section.ipc-page-section.ipc-page-section--base.sc-cd7dc4b7-0.ycheS.title-cast
.title-cast--movie.celwidget")
    private WebElement castSection;

    @FindBy(css = "div.sc-cd7dc4b7-7.vCane > a")
    private WebElement castSectionActor;

    @FindBy(css = "div.ipc-starbar__rating > button:nth-child(9)")
    private WebElement rateNineStars;

    @FindBy(css = ".ipc-text-button.ipc-rating-prompt__secondary-button > span")
    private WebElement removeRatingButton;

    @FindBy(css = "[data-testid='tm-box-wl-button']")
    private WebElement addTitleToWatchlist;

    @FindBy(css = "div.ipc-rating-prompt__rating-container > button")
    private WebElement sumbitRatingButton;

    public String getTitleName() {
        return titleName.getText();
    }

    public String getOriginalTitleName() {
        return originalTitleName.getText();
```

```
}

public String getTitleDetails() {
    return titleDetails.getText();
}

public String getTitleRating() {
    return titleRating.getText();
}

public String getTitleUserRating() {
    return titleUserRating.getText();
}

public WebElement getCastSection() {
    return castSection;
}

public WebElement getTitleUserRatingElement() {
    return titleUserRating;
}

public void rateTitle() {
    rateButton.click();
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(ExpectedConditions.elementToBeClickable(rateNineStars));
    JavascriptExecutor js = (JavascriptExecutor) driver;
    js.executeScript("arguments[0].click();", rateNineStars);
    implicitWait(500);
    sumbitRatingButton.click();
}

public void removeTitleRating() {
    rateButton.click();
    removeRatingButton.click();
}

public void getCastSectionActor() {
```

```
        scrollToElement(castSectionActor);

        implicitWait(1000);

        castSectionActor.click();
    }

    public void addTitleToWatchList() {
        scrollToElement(addTitleToWatchlist);

        implicitWait(500);

        addTitleToWatchlist.click();
    }
}
```

WatchlistPage class

```
public class WatchlistPage extends BasePage {
    public WatchlistPage(WebDriver driver) {
        super(driver);
    }

    @FindBy(css = "[data-testid = 'list-page-atf-add-to-list-btn']")
    private WebElement createNewListButton;

    @FindBy(css = "[data-testid = 'input-list-name']")
    private WebElement enterListNameField;

    @FindBy(css = "[data-testid = 'input-list-description']")
    private WebElement enterListDescriptionField;

    @FindBy(css = "[data-testid = 'list-create-button']")
    private WebElement createListButton;

                                                @FindBy(css                    =
"div.ipc-title.ipc-title--base.ipc-title--title.ipc-title-link-no-icon.ipc-titl
e--on-textPrimary.sc-a69a4297-2.bqNXEn.dli-title.with-margin > a > h3")
    private WebElement watchlistTitle;
```

```
                                                    @FindBy(css =
"div.sc-65d2a03-1.hLElui.ipc-page-grid__item.ipc-page-grid__item--span-2")
    private WebElement watchlistBody;

    public WebElement getWatchlistTitle() {
        return watchlistTitle;
    }

    public void createNewList(String listName, String listDescription) {
        createNewListButton.click();

        enterListNameField.sendKeys(listName);
        enterListDescriptionField.sendKeys(listDescription);

        implicitWait(2000);

        createListButton.click();

        implicitWait(2000);
    }

    public WebElement getWatchlistBody() {
        return watchlistBody;
    }

    public String getWatchlistTitleText() {
        return watchlistTitle.getText();
    }

    public String getWatchlistBodyText() {
        return watchlistBody.getText();
    }

}
```

2.3.3 DriverManager Class

```
public class DriverManager {
    private static WebDriver driver;

    public static WebDriver getDriver() {
        if (driver == null) {
            try {
                WebDriverManager.chromedriver().setup();
                driver = new ChromeDriver();
            } catch (Exception e) {
                        System.err.println("Failed to initialize WebDriver: " +
e.getMessage());
                e.printStackTrace();
                throw new RuntimeException("WebDriver setup failed", e);
            }
        }
        return driver;
    }

    public static void quitDriver() {
        if (driver != null) {
            try {
                driver.quit();
            } catch (Exception e) {
                            System.err.println("Failed to quit WebDriver: " +
e.getMessage());
                e.printStackTrace();
            } finally {
                driver = null;
            }
        }
    }
}
```

2.3.4 Config Class

```
public class Config {
    public static final String BASE_URL = "https://www.imdb.com/";
    public static final String EMAIL = "lotrxbot@gmail.com";
    public static final String PASSWORD = "Ganbatte123";
}
```

3. Test Execution

Test execution is performed by running the TestSuite class which runs tests in a logical order. For example, by running SignInTest before other tests we can ensure that cookies are saved if the sign in was successful, and therefore avoid the sign in process in tests that require user to be signed in. Every test class inherits BaseTest class which handles SetUp, TearDown and other helper methods.

BaseTest Class:

```
public class BaseTest {

    protected WebDriver driver;


    @BeforeEach
    public void setUp() {
        driver = DriverManager.getDriver();
        driver.manage().window().maximize();
        driver.get(Config.BASE_URL);
    }


    @AfterEach
    public void tearDown() {
        DriverManager.quitDriver();
    }
```

```
void loadCookies() {
    try (FileInputStream fileIn = new FileInputStream("cookies.pkl");
         ObjectInputStream in = new ObjectInputStream(fileIn)) {
        Set<Cookie> cookies = (Set<Cookie>) in.readObject();
        for (Cookie cookie : cookies) {
            driver.manage().addCookie(cookie);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    driver.navigate().refresh();
}

public void signInOrLoadCookies() {
    File cookiesFile = new File("cookies.pkl");
    if (cookiesFile.exists()) {
        loadCookies();
    } else {
        SignInPage signInPage = new SignInPage(driver);
        signInPage.signIn(Config.EMAIL, Config.PASSWORD);
        saveCookies();
    }
}

void saveCookies() {
    Set<Cookie> cookies = driver.manage().getCookies();
    try (FileOutputStream fileOut = new FileOutputStream("cookies.pkl");
         ObjectOutputStream out = new ObjectOutputStream(fileOut)) {
```

```
            out.writeObject(cookies);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    protected void implicitWait(int ms) {
        try {
            Thread.sleep(ms);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            System.err.println("Sleep interrupted: " + e.getMessage());
        }
    }
}
```

Tests are run in this order:

User Authentication ➡ UI Tests ➡ Localization Test ➡ Functionality Tests ➡ Performance Tests

3.1. Sign In Test

Test Name: signInTestSuccess				
Description: Check if the user is able to sign in with provided credentials and lastly save cookies if the test was successful.				
Pre-condition(s): Credentials provided in tests have to match credentials of previously registered user.				
Test Steps: 1. Click on Navigation Bar Sign In button. 2. Click on Sign In with IMDb account button. 3. Type in email address and password. 4. Click on submit button. 5. Check if successful sign in reference is shown in URL.	**Test Data:** Email address: lotrxbot@gmail.com Password: Ganbatte123	**Expected Result:** The user is successfully signed in.	**Actual Result:** The user is successfully signed in.	**Status:** PASS
Notes: Bot check could be triggered and test automation would not work.				

```
@Order(1)
@Test
public void signInTestSuccess() {
    SignInPage signInPage = new SignInPage(driver);
    signInPage.signIn(Config.EMAIL, Config.PASSWORD);

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));

    Assertions.assertEquals("https://www.imdb.com/?ref_=login",
driver.getCurrentUrl(), "Login was unsuccessful!");

    saveCookies();
}
```

Test Name: signInEmptyFieldsTestFailure
Description: Check if an error is shown if user tries to sign in without entering credentials.
Pre-condition(s): Credentials provided in tests have to match credentials of previously registered user.

Test Steps:	Test Data:	Expected Result:	Actual Result:	Status:
1. Click on Navigation Bar Sign In button. 2. Click on Sign In with IMDb account button. 3. Leave email address and password fields empty. 4. Click on submit button. 5. Check if error is shown.		The error is shown and user was unable to sign in.	The error is shown and user was unable to sign in.	PASS
Notes:				

```
@Order(2)
@Test
public void signInEmptyFieldsTestFailure() {
   SignInPage signInPage = new SignInPage(driver);

   signInPage.signIn("", "");

   WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
   wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));

   Assertions.assertNotEquals("https://www.imdb.com/?ref_=login",
driver.getCurrentUrl(), "Login was successful!");

   Assertions.assertTrue(signInPage.getAuthErrorMsgBox().isDisplayed(),
"Authentication error message not shown!");
}
```

Test Name: signInTestFailure				
Description: Check if the user isn't able to log in when invalid credentials are entered.				
Pre-condition(s): Credentials provided in tests have to be invalid.				
Test Steps:	**Test Data:**	**Expected Result:**	**Actual Result:**	**Status:**
1. Click on Navigation Bar Sign In button. 2. Click on Sign In with IMDb account button. 3. Enter invalid credentials.	Email address: wrongemail@gmail.com Password: wrongpassword	User was unable to sign in.	User was unable to sign in.	PASS

4. Click on submit button. 5. Check if sign in was unsuccessful.				
Notes:				

```
@Order(3)
@Test
public void signInTestFailure() {
   SignInPage signInPage = new SignInPage(driver);

   signInPage.signIn("wrongemail@gmail.com", "wrongpassword");

   WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
   wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));

   Assertions.assertNotEquals("https://www.imdb.com/?ref_=login",
driver.getCurrentUrl(), "Login was successful!");
}
```

3.2. Sign Up Test

Test Name: signUpTestSuccess				
Description: Check if the user is able to sign up with provided credentials.				
Pre-condition(s): There isn't an already registered account with provided credentials.				
Test Steps: 1. Click on Navigation Bar Sign In button. 2. Click on Create New Account button. 3. Enter credentials. 4. Click on Sign Up button. 5. Check if the Sign Up passed and the request for confirmation code is shown.	**Test Data:** Username: Toge Inumaki Email address: lot.rxbot@gmail.com Password: BonitoFlakes123	**Expected Result:** Sign Up has passed and the request for confirmation code is shown.	**Actual Result:** Sign Up has passed and the request for confirmation code is shown.	**Status:** PASS
Notes: Bot check could be triggered and test automation would not work.				

```
@Test
public void signUpTestSuccess() {
    SignInPage signInPage = new SignInPage(driver);

    signInPage.signUp("Toge Inumaki", "lot.rxbot@gmail.com", "BonitoFlakes123");

    implicitWait(1000);


Assertions.assertTrue(driver.getCurrentUrl().contains("https://www.imdb.com/ap/
cvf/request"), "Sign up unsuccessful!");
}
```

Test Name: signUpInvalidEmailTest				
Description: Check if error is shown if user tries to sign up with invalid email address.				
Pre-condition(s): There isn't an already registered account with provided credentials.				
Test Steps: 1. Click on Navigation Bar Sign In button. 2. Click on Create New Account button.	**Test Data:** Username: Toge Inumaki Email address: invalidemailinput	**Expected Result:** Error was shown.	**Actual Result:** Error was shown.	**Status:** PASS

3. Enter credentials with wrongly formatted email address. 4. Click on Sign Up button. 5. Check if error is shown.	Password: BonitoFlakes123			
Notes:				

```
@Test
public void signUpInvalidEmailTest() {
   SignInPage signInPage = new SignInPage(driver);

   signInPage.signUp("Toge Inumaki", "invalidemailinput", "BonitoFlakes123");

   Assertions.assertTrue(signInPage.getInvalidEmailInput().isDisplayed(),
"Invalid email was accepted!");
}
```

Test Name: signUpInvalidPasswordTest				
Description: Check if an error is shown if user tries to sign up with invalid password.				
Pre-condition(s): There isn't an already registered account with provided credentials.				
Test Steps: 1. Click on Navigation Bar Sign In button. 2. Click on Create New Account button. 3. Enter credentials with wrongly formatted password. 4. Click on Sign Up button. 5. Check if error is shown.	**Test Data:** Username: Toge Inumaki Email address: realemail@gmail.com Password: gomen	**Expected Result:** Error was shown.	**Actual Result:** Error was shown.	**Status:** PASS
Notes:				

3.3. Guest User Navigation Bar Test

Test Name: logoIconTestRef				
Description: Check if the user is redirected to the home page when clicks on navigation bar logo icon.				
Pre-condition(s): User must not be logged in.				
Test Steps: 1. Click on navigation bar IMDb logo icon. 2. Check if user is redirected to the home page.	**Test Data:**	**Expected Result:** User was redirected to the home page.	**Actual Result:** User was redirected to the home page.	**Status:** PASS
Notes:				

```
@Test
public void logoIconTestRef() {
   HomePage homePage = new HomePage(driver);

   homePage.getNavBarLogo().click();

   Assertions.assertEquals("https://www.imdb.com/?ref_=nv_home",
driver.getCurrentUrl(), "Clicking on the logo icon doesn't redirect to the home
page!");
}
```

Test Name: imdbProIconTestRef				
Description: Check if user is redirected to the IMDb pro sign in page when clicks on navigation bar IMDb pro icon.				
Pre-condition(s): User must not be logged in.				
Test Steps: 1. Click on navigation bar IMDb logo icon. 2. Check if user is redirected to the home page.	**Test Data:**	**Expected Result:** User was redirected to the home page.	**Actual Result:** User was redirected to the home page.	**Status:** PASS
Notes:				

```
@Test
public void imdbProIconTestRef() {
   HomePage homePage = new HomePage(driver);
```

```
   String expectedUrl = "^https:\\/\\/pro\\.imdb\\.com\\/login.*";

   homePage.getNavBarIMDBProIcon().click();

   Assertions.assertTrue(driver.getCurrentUrl().matches(expectedUrl), "Clicking
on the imdb pro icon doesn't redirect to pro sign in page!");
}
```

Test Name: watchlistIconTestRef				
Description: Check if the user is redirected to the sign in page when clicks on navigation bar watchlist icon.				
Pre-condition(s): User must not be logged in.				
Test Steps: 1. Click on navigation bar watchlist icon. 2. Check if user is redirected to the sign in page.	**Test Data:**	**Expected Result:** User was redirected to the sign in page.	**Actual Result:** User was redirected to the sign in page.	**Status:** PASS
Notes:				

```
@Test
public void watchlistIconTestRef() {
   HomePage homePage = new HomePage(driver);
   String expectedURL =
"^https:\\/\\/www\\.imdb\\.com\\/registration\\/signin.*";

   homePage.getNavBarWatchlistButton().click();

   Assertions.assertTrue(driver.getCurrentUrl().matches(expectedURL),
"Watchlist icon doesn't work!");
}
```

Test Name: signInIconTestRef				
Description: Check if user is redirected to the sign in page when clicks on navigation bar sign in icon.				
Pre-condition(s): User must not be logged in.				
Test Steps: 1. Click on navigation bar sign in icon.	**Test Data:**	**Expected Result:** User was	**Actual Result:** User was redirected to the	**Status:** PASS

2. Check if user is redirected to the sign in page.		redirected to the sign in page.	sign in page.	
Notes:				

```
@Test
public void signInIconTestRef() {
   HomePage homePage = new HomePage(driver);
   String expectedURL =
"^https:\\/\\/www\\.imdb\\.com\\/registration\\/signin.*";

   homePage.getNavBarSignInButton().click();

   Assertions.assertTrue(driver.getCurrentUrl().matches(expectedURL), "Sign in
icon doesn't work!");
}
```

Test Name: menuShownTest				
Description: Check if menu is shown when user clicks on navigation bar menu icon.				
Pre-condition(s): User must not be logged in.				
Test Steps: 1. Click on navigation bar menu icon. 2. Check if menu is shown.	**Test Data:**	**Expected Result:** Menu is shown.	**Actual Result:** Menu is shown.	**Status:** PASS
Notes:				

```
@Test
public void menuShownTest() {
   HomePage homePage = new HomePage(driver);

   homePage.getNavBarMenu().click();

   WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
   WebElement navBarMenu =
wait.until(ExpectedConditions.visibilityOf(homePage.getNavBarMenuShown()));

   Assertions.assertTrue(navBarMenu.isDisplayed(), "Menu not shown!");
}
```

Test Name: languageMenuShownTest				
Description: Check if language menu is shown when user clicks on navigation bar language menu icon.				
Pre-condition(s): User must not be logged in.				
Test Steps: 1. Click on navigation bar language menu icon. 2. Check if language menu is shown.	**Test Data:**	**Expected Result:** Language menu is shown.	**Actual Result:** Language menu is shown.	**Status:** PASS
Notes:				

```
@Test
public void languageMenuShownTest() {
   HomePage homePage = new HomePage(driver);

   homePage.getNavBarLanguageSelectorButton().click();

   WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
   WebElement navBarLanguageMenu =
wait.until(ExpectedConditions.visibilityOf(homePage.getNavBarLanguageMenu()));

   Assertions.assertTrue(navBarLanguageMenu.isDisplayed(), "Language menu not
shown!");
}
```

Test Name: findPageToHomePageTest				
Description: Check if user is redirected back to the home page when clicks on IMDb logo while being on another page, in this case find page.				
Pre-condition(s): User must not be logged in.				
Test Steps: 1. Enter title name in a search bar. 2. Click on search button. 3. Click on IMDb logo icon. 4. Check if user is	**Test Data:** Title name to be searched: parasite	**Expected Result:** User is redirected back to the home page.	**Actual Result:** User is redirected back to the home page.	**Status:** PASS

redirected back to the home page.				
Notes:				

```
@Test
public void findPageToHomePageTest() {
   HomePage homePage = new HomePage(driver);
   FindPage findPage = new FindPage(driver);

   homePage.searchForText("parasite");

   WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
   wait.until(ExpectedConditions.visibilityOf(findPage.getTitleSection()));

   homePage.getNavBarLogo().click();

   wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));

   Assertions.assertEquals("https://www.imdb.com/?ref_=nv_home",
driver.getCurrentUrl(), "Navigation from find page to home page through imdb
logo icon is not correct!");
}
```

3.4. Registered User Navigation Bar Test

Test Name: logoIconTestRef				
Description: Check if user is redirected to the home page when clicks on navigation bar logo icon.				
Pre-condition(s): User must be logged in.				
Test Steps: 1. Click on navigation bar IMDb logo icon. 2. Check if user is redirected to the home page.	**Test Data:**	**Expected Result:** User was redirected to the home page.	**Actual Result:** User was redirected to the home page.	**Status:** PASS
Notes:				

```
@Test
public void logoIconTestRef() {
```

```
   HomePage homePage = new HomePage(driver);

   homePage.getNavBarLogo().click();

   Assertions.assertEquals("https://www.imdb.com/?ref_=nv_home",
driver.getCurrentUrl(), "Clicking on the logo icon doesn't redirect to the home
page!");
}
```

Test Name: imdbProIconTestRef				
Description: Check if user is redirected to the IMDb pro sign in page when clicks on navigation bar IMDb pro icon.				
Pre-condition(s): User must be logged in.				
Test Steps: 1. Click on navigation bar IMDb logo icon. 2. Check if user is redirected to the home page.	**Test Data:**	**Expected Result:** User was redirected to the home page.	**Actual Result:** User was redirected to the home page.	**Status:** PASS
Notes:				

```
@Test
public void imdbProIconTestRef() {
   HomePage homePage = new HomePage(driver);
   String expectedUrl = "^https:\\/\\/pro\\.imdb\\.com\\/login.*";

   homePage.getNavBarIMDBProIcon().click();

   Assertions.assertTrue(driver.getCurrentUrl().matches(expectedUrl), "Clicking
on the imdb pro icon doesn't redirect to pro sign in page!");
}
```

Test Name: watchlistIconTestRef				
Description: Check if user is redirected to the watchlist page when clicks on navigation bar watchlist icon.				
Pre-condition(s): User must be logged in.				
Test Steps: 1. Click on navigation bar watchlist icon.	**Test Data:**	**Expected Result:** User was	**Actual Result:** User was redirected to the	**Status:** PASS

2. Check if user is redirected to the watchlist page.		redirected to the watchlist page.	watchlist page.	
Notes:				

```
@Test
public void watchlistIconTestRef() {
   HomePage homePage = new HomePage(driver);
   String expectedURL =
"^https:\\/\\/www\\.imdb\\.com\\/user\\/ur[0-9a-zA-Z]+\\/watchlist\\/$";

   homePage.getNavBarWatchlistButton().click();

   Assertions.assertTrue(driver.getCurrentUrl().matches(expectedURL), "User not
redirected to the watchlist page!");
}
```

Test Name: navBarProfileIconTest				
Description: Check if user is redirected to the profile page when clicks on navigation bar profile icon.				
Pre-condition(s): User must be logged in.				
Test Steps: 1. Click on navigation bar profile icon. 2. Check if user is redirected to the profile page.	**Test Data:**	**Expected Result:** User was redirected to the profile page.	**Actual Result:** User was redirected to the profile page.	**Status:** PASS
Notes:				

```
@Test
public void navBarProfileIconTest() {
   HomePage homePage = new HomePage(driver);
   String expectedUrl = "^https:\\/\\/www\\.imdb\\.com\\/user\\/ur.+$";

   homePage.getNavBarProfileIcon().click();

   WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
   WebElement navBarProfileMenu =
wait.until(ExpectedConditions.visibilityOf(homePage.getNavBarProfileMenu()));

   Assertions.assertTrue(navBarProfileMenu.isDisplayed(), "Nav bar profile menu
not displayed if profile icon is clicked!");

   implicitWait(500);
```

```
    homePage.getNavBarProfileMenuProfile().click();

    Assertions.assertTrue(driver.getCurrentUrl().matches(expectedUrl), "Not
navigated to user profile!");
}
```

Test Name: menuShownTest				
Description: Check if menu is shown when user clicks on navigation bar menu icon.				
Pre-condition(s): User must not be logged in.				
Test Steps: 1. Click on navigation bar menu icon. 2. Check if menu is shown.	**Test Data:**	**Expected Result:** Menu is shown.	**Actual Result:** Menu is shown.	**Status:** PASS
Notes:				

```
@Test
public void menuShownTest() {
    HomePage homePage = new HomePage(driver);

    homePage.getNavBarMenu().click();

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    WebElement navBarMenu =
wait.until(ExpectedConditions.visibilityOf(homePage.getNavBarMenuShown()));

    Assertions.assertTrue(navBarMenu.isDisplayed(), "Menu not shown!");
}
```

Test Name: languageMenuShownTest				
Description: Check if language menu is shown when user clicks on navigation bar language menu icon.				
Pre-condition(s): User must be logged in.				
Test Steps: 1. Click on navigation bar language menu icon. 2. Check if language	**Test Data:**	**Expected Result:** Language menu is shown.	**Actual Result:** Language menu is shown.	**Status:** PASS

menu is shown.				
Notes:				

```
@Test
public void languageMenuShownTest() {
   HomePage homePage = new HomePage(driver);

   homePage.getNavBarLanguageSelectorButton().click();

   WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
   WebElement navBarLanguageMenu =
wait.until(ExpectedConditions.visibilityOf(homePage.getNavBarLanguageMenu()));

   Assertions.assertTrue(navBarLanguageMenu.isDisplayed(), "Language menu not
shown!");
}
```

Test Name: findPageToHomePageTest				
Description: Check if user is redirected back to the home page when clicks on IMDb logo while being on another page, in this case find page.				
Pre-condition(s): User must be logged in.				
Test Steps: 1. Enter title name in a search bar. 2. Click on search button. 3. Click on IMDb logo icon. 4. Check if user is redirected back to the home page.	**Test Data:** Title name to be searched: parasite	**Expected Result:** User is redirected back to the home page.	**Actual Result:** User is redirected back to the home page.	**Status:** PASS
Notes:				

```
@Test
public void findPageToHomePageTest() {
   HomePage homePage = new HomePage(driver);
   FindPage findPage = new FindPage(driver);

   homePage.searchForText("parasite");

   WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
   wait.until(ExpectedConditions.visibilityOf(findPage.getTitleSection()));

   homePage.getNavBarLogo().click();
```

```
   wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));

   Assertions.assertEquals("https://www.imdb.com/?ref_=nv_home",
driver.getCurrentUrl(), "Navigation from find page to home page through imdb
logo icon is not correct!");
}
```

3.5. Title Page Test

Test Name: findTitlePageTest				
Description: Check if user is able to successfully search for a title and is redirected to the title page.				
Pre-condition(s):				
Test Steps: 1. Enter title name in a search bar. 2. Click on search	**Test Data:** Title name to be searched: Nobody Knows	**Expected Result:** User can successfully	**Actual Result:** User can successfully search for a	**Status:** PASS

button. 3. Click on the title that comes up first. 4. Check if user is redirected to the page of a searched title.		search for a movie and is redirected to the right title page.	movie and is redirected to the right title page.	
Notes:				

```
@Test
public void findTitlePageTest() {
    FindPage findPage = new FindPage(driver);

    findPage.FindTitle("Nobody Knows");

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));


Assertions.assertEquals("https://www.imdb.com/title/tt0408664/?ref_=fn_all_ttl_
1", driver.getCurrentUrl(), "Not redirected to title page!");
}
```

Test Name: titleDetailsTest				
Description: Check if details of the title on the title page are correct.				
Pre-condition(s):				
Test Steps: 1. Enter title name in a search bar. 2. Click on search button. 3. Click on the title that comes up first. 4. Check if the title name on the page loaded is correct. 5. Check if the original title name on the page loaded is correct. 6. Check if the other details on the page	**Test Data:** Title name to be searched: Shoplifters	**Expected Result:** Details of the title on the title page are correct.	**Actual Result:** Details of the title on the title page are correct.	**Status:** PASS

loaded are correct. 7. Check if title rating is shown. 8. Check if cast cestion is displayed.				
Notes:				

```
@Test
public void titleDetailsTest() {
   FindPage findPage = new FindPage(driver);
   TitlePage titlePage = new TitlePage(driver);

   findPage.FindTitle("Shoplifters");

   WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
   wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));

   Assertions.assertEquals("Shoplifters" ,titlePage.getTitleName(), "Title name
doesn't match!");
   Assertions.assertTrue(titlePage.getOriginalTitleName().contains("Manbiki
kazoku"));
   String[] titleDetails = {"2018", "15", "2h 1m"};
   for(String detail : titleDetails) {
       Assertions.assertTrue(titlePage.getTitleDetails().contains(detail));
   }
   Assertions.assertTrue(titlePage.getTitleRating().contains("/10"), "Title
rating is not displayed!");
   Assertions.assertTrue(titlePage.getCastSection().isDisplayed(), "Cast not
displayed!");
}
```

Test Name: getActorTest				
Description: Check if user is able to successfully search for an actor on the title page.				
Pre-condition(s):				
Test Steps: 1. Enter title name in a search bar. 2. Click on search button. 3. Click on the title that comes up first. 4. Scroll to the actor	**Test Data:** Title name to be searched: Mean Girls	**Expected Result:** User is successfully redirected to the correct page of an actor.	**Actual Result:** User is successfully redirected to the correct page of an actor.	**Status:** PASS

section. 5. Click on a specified actor. 6. Check if user is redirected to the correct page of an actor.				
Notes:				

```
@Test
public void getActorTest() {
    FindPage findPage = new FindPage(driver);
    TitlePage titlePage = new TitlePage(driver);

    findPage.FindTitle("Mean Girls");

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));

    titlePage.getCastSectionActor();

    wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));


Assertions.assertEquals("https://www.imdb.com/name/nm0517820/?ref_=tt_cst_t_1",
driver.getCurrentUrl(), "Not redirected to actor page!");
}
```

3.6. Localization Test

Test Name: testSwitchLanguageGerman				
Description: Check if language is successfully changed to German when done so.				
Pre-condition(s):				
Test Steps: 1. Click on the language options navigation bar menu. 2. Click on German option. 3. Check if URL has	**Test Data:**	**Expected Result:** Language is successfully changed to German.	**Actual Result:** Language is successfully changed to German.	**Status:** PASS

changed correctly. 4. Check if Sign In button has been translated correctly. 5. Check if language selector code is correct.				
Notes:				

Test Name: testSwitchLanguageItalian				
Description: Check if language is successfully changed to Italian when done so.				
Pre-condition(s):				
Test Steps: 1. Click on the language options navigation bar menu. 2. Click on Italian option. 3. Check if URL has changed correctly. 4. Check if Sign In button has been translated correctly. 5. Check if language selector code is correct.	**Test Data:**	**Expected Result:** Language is successfully changed to Italian.	**Actual Result:** Language is successfully changed to Italian.	**Status:** PASS
Notes:				

Test Name: testSwitchLanguageEnglish				
Description: Check if language is successfully changed to English when done so.				
Pre-condition(s):				
Test Steps: 1. Click on the language options navigation bar menu. 2. Click on English option.	**Test Data:**	**Expected Result:** Language is successfully changed to English.	**Actual Result:** Language is successfully changed to English.	**Status:** PASS

3. Check if URL has changed correctly. 4. Check if Sign In button has been translated correctly. 5. Check if language selector code is correct.				
Notes:				

```
@ParameterizedTest
@CsvSource({
        "German, https://www.imdb.com/de/, Anmelden, DE",
        "Italian, https://www.imdb.com/it/, Accedi, IT",
        "English, https://www.imdb.com/, Sign In, EN"
})
void testSwitchLanguage(String language, String expectedUrl, String
expectedSignInText, String expectedLanguageCode) {
    HomePage homePage = new HomePage(driver);
    homePage.switchLanguage(language);

    Assertions.assertEquals(expectedUrl, driver.getCurrentUrl(), "URL not
changed correctly!");
    Assertions.assertEquals(expectedSignInText,
homePage.getNavBarSignInButton().getText(), "Sign in not translated
correctly!");
    Assertions.assertEquals(expectedLanguageCode,
homePage.getNavBarLanguageSelectorButton().getText(), "Language selector code
mismatch!");
}
```

Test Name: backToEnglish				
Description: Check if language is successfully changed back to English when first changed to another language, in this case Italian.				
Pre-condition(s):				
Test Steps: 1. Click on the language options navigation bar menu. 2. Click on Italian	**Test Data:**	**Expected Result:** Language is successfully changed back to	**Actual Result:** Language is successfully changed back to English.	**Status:** PASS

option. 3. Click on the language options navigation bar menu. 4. Click on English option. 5. Check if URL has changed correctly. 6. Check if Sign In button has been translated correctly. 7. Check if language selector code is correct.		English.		
Notes:				

```
@Test
void backToEnglish() {
   HomePage homePage = new HomePage(driver);

   homePage.switchLanguage("Italian");

   homePage.switchLanguage("English");

   WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
   wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));


   Assertions.assertEquals("https://www.imdb.com/", driver.getCurrentUrl(),
"Url not changed to English!");
   Assertions.assertEquals("Sign In",
homePage.getNavBarSignInButton().getText(), "Sign in not translated
correctly!");
   Assertions.assertEquals("EN",
homePage.getNavBarLanguageSelectorButton().getText(), "Language code not set to
IT!");
}
```

3.7. Watchlist Test

Test Name: addToWatchlistTest
Description: Check if user is able to add a title to his watchlist.
Pre-condition(s): User must be logged in. The specified title must not be already added to the watchlist.

Test Steps:	Test Data:	Expected Result:	Actual Result:	Status:
1. Enter the title name in a navigation bar search bar. 2. Click on search button. 3. Click on the first title that comes up. 4. Scroll to the Add to Watchlist button. 5. Click the Add to Watchlist button. 6. Navigate to watchlist page. 7. Check if movie is added to the watchlist.	Title name to be searched: Django Unchained	User is able to add a title to his watchlist.	User is able to add a title to his watchlist.	PASS
Notes:				

```
@Test
public void addToWatchlistTest() {
    FindPage findPage = new FindPage(driver);
    TitlePage titlePage = new TitlePage(driver);
    WatchlistPage watchlistPage = new WatchlistPage(driver);

    findPage.FindTitle("Django Unchained");

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
     wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));

    implicitWait(2000);

    titlePage.addTitleToWatchList();

    implicitWait(2000);

    driver.navigate().to("https://www.imdb.com/list/watchlist");
```

```
    implicitWait(2000);

    Assertions.assertTrue(watchlistPage.getWatchlistTitleText().contains("Django
Unchained"), "Couldn't add title to the watchlist!");
}
```

Test Name: removeFromWatchlistTest				
Description: Check if user is able to remove a title from his watchlist.				
Pre-condition(s): User must be logged in. The specified title must be already added to the watchlist.				
Test Steps: 1. Enter the title name in a navigation bar search bar. 2. Click on search button. 3. Click on the first title that comes up. 4. Scroll to the Add to Watchlist button. 5. Click the Add to Watchlist button. 6. Navigate to watchlist page. 7. Check if movie is removed from the watchlist.	**Test Data:** Title name to be searched: Django Unchained	**Expected Result:** User is able to remove a title from his watchlist.	**Actual Result:** User is able to remove a title from his watchlist.	**Status:** PASS
Notes:				

```
@Test
public void removeFromWatchlistTest() {
    FindPage findPage = new FindPage(driver);
    TitlePage titlePage = new TitlePage(driver);
    WatchlistPage watchlistPage = new WatchlistPage(driver);

    findPage.FindTitle("Django Unchained");
```

```
   WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
     wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));

   implicitWait(2000);

   titlePage.addTitleToWatchList();

   implicitWait(2000);

   driver.navigate().to("https://www.imdb.com/list/watchlist");


wait.until(ExpectedConditions.elementToBeClickable(watchlistPage.getWatchlistBo
dy()));

       Assertions.assertTrue(watchlistPage.getWatchlistBodyText().contains("This
list is empty."), "Couldn't remove title from a watchlist!");
}
```

Test Name: createNewListTest				
Description: Check if user is able to create a custom list.				
Pre-condition(s): User must be logged in.				
Test Steps: 1. Click on the navigation bar watchlist button. 2. Click on create a new list button. 3. Enter the list name and description. 4. Click on the create list button. 5. Check if list was created.	**Test Data:**	**Expected Result:** User is able to create a custom list.	**Actual Result:** User is able to create a custom list.	**Status:** PASS
Notes:				

```
@Test
public void createNewListTest() {
   HomePage homePage = new HomePage(driver);
   WatchlistPage watchlistPage = new WatchlistPage(driver);

   homePage.getNavBarWatchlistButton().click();

   WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));

    watchlistPage.createNewList("Japanese movies list", "List of my favorite
Japanese movies!");

    wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));

     Assertions.assertTrue(driver.getCurrentUrl().contains("ref_=cr_lst_crte"),
"New list not created!");
}
```

3.8. Recently Viewed List Test

Test Name: recentlyViewedTest				
Description: Check if user is able to see a list of recently viewed titles.				
Pre-condition(s): User must be logged in.				
Test Steps: 1. Enter the title name in a navigation bar search bar. 2. Click on search	**Test Data:** Title name to be searched: Breaking Bad	**Expected Result:** User is able to see a recently viewed list.	**Actual Result:** User is able to see a recently viewed list.	**Status:** PASS

button. 3. Click on the first title that comes up. 4. Navigate to the home page. 5. Scroll to the recently viewed section. 6. Check if the title is shown in the recently viewed section.				
Notes:				

```
@Test
void recentlyViewedTest() {
   HomePage homePage = new HomePage(driver);
   FindPage findPage = new FindPage(driver);

   findPage.FindTitle("Breaking Bad");

   implicitWait(2000);

   driver.navigate().to("https://www.imdb.com");

   implicitWait(2000);

   homePage.goToRecentlyViewedSection();

   implicitWait(2000);


Assertions.assertTrue(homePage.getRecentlyViewedSection().getText().contains("B
reaking Bad"), "Recently viewed functionality doesn't work!");
}
```

Test Name: clearRecentlyViewedTest
Description: Check if user is able to clear a list of recently viewed titles.

Pre-condition(s): User must be logged in.				
Test Steps: 1. Enter the title name in a navigation bar search bar. 2. Click on search button. 3. Click on the first title that comes up. 4. Navigate to the home page. 5. Scroll to the recently viewed section. 6. Click on a clear recently viewed button. 7. Check if recently viewed section is not displayed.	**Test Data:** Title name to be searched: Abbas Kiarostami	**Expected Result:** User is able to cleara recently viewed list.	**Actual Result:** User is able to cleara recently viewed list.	**Status:** PASS
Notes:				

```
@Test
void clearRecentlyViewedTest() {
    HomePage homePage = new HomePage(driver);
    FindPage findPage = new FindPage(driver);

    findPage.FindTitle("Abbas Kiarostami");

    implicitWait(2000);

    driver.navigate().to("https://www.imdb.com");

    implicitWait(2000);

    homePage.goToRecentlyViewedSection();

    implicitWait(2000);

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
```

```
wait.until(ExpectedConditions.elementToBeClickable(homePage.getClearRecentlyVie
wed()));

   homePage.goToRecentlyViewedSection();

   implicitWait(2000);

          ((JavascriptExecutor)    driver).executeScript("arguments[0].click();",
homePage.getClearRecentlyViewed());

   implicitWait(2000);

              Assertions.assertThrows(NoSuchElementException.class,     ()     ->
{homePage.getRecentlyViewedSection().isDisplayed();},    "Couldn't    clear    the
recently viewed list!");
}
```

3.9. Search Test

Test Name: searchTitleSuccess				
Description: Check if user is able to successfully search for a title and is redirected to the title page.				
Pre-condition(s):				
Test Steps:	**Test Data:**	**Expected Result:**	**Actual Result:**	**Status:**

1. Enter title name in a search bar. 2. Click on search button. 3. Check if title user is looking for is shown.	Title name to be searched: parasite	User can successfully perform a search.	User can successfully perform a search.	PASS
Notes:				

```
@Test
public void searchTitleSuccess() {
   HomePage homePage = new HomePage(driver);
   FindPage findPage = new FindPage(driver);

   homePage.searchForText("parasite");

   WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
   WebElement titleSection =
wait.until(ExpectedConditions.visibilityOf(findPage.getTitleSection()));

   Assertions.assertTrue(titleSection.getText().contains("Parasite"), "Search
result does not include 'Parasite'");
}
```

Test Name: searchTitleFailure				
Description: Check if no results are found after searching for nonexistent title.				
Pre-condition(s):				
Test Steps: 1. Enter a nonexistent title name in a search bar. 2. Click on search button. 3. Check if “no results found” message is shown.	**Test Data:** Title name to be searched: some nonexistent movie	**Expected Result:** “no results found” message is shown.	**Actual Result:** “no results found” message is shown.	**Status:** PASS
Notes:				

```
@Test
public void searchTitleFailure() {
   HomePage homePage = new HomePage(driver);
   FindPage findPage = new FindPage(driver);

   homePage.searchForText("some nonexistent movie");
```

```
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    WebElement firstResult =
wait.until(ExpectedConditions.visibilityOf(findPage.getTitleSection()));

    Assertions.assertTrue(findPage.getTitleSection().isDisplayed());
    Assertions.assertTrue(firstResult.getText().contains("No results found for
\"some nonexisting movie\""), "Test for invalid title search failed.");
}
```

Test Name: searchEmptyField				
Description: Check if no sections are shown when searching for an empty field.				
Pre-condition(s):				
Test Steps: 1. Enter a nothing in a search bar. 2. Click on search button. 3. Check if sections are not shown.	**Test Data:**	**Expected Result:** No sections are shown.	**Actual Result:** No sections are shown.	**Status:** PASS
Notes:				

```
@Test
public void searchEmptyField() {
    HomePage homePage = new HomePage(driver);
    FindPage findPage = new FindPage(driver);

    homePage.searchForText(" ");

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));

    Assertions.assertThrows(NoSuchElementException.class ,() ->
{findPage.getInterestSection().isDisplayed();}, "Interest section is
displayed!");
    Assertions.assertThrows(NoSuchElementException.class ,() ->
{findPage.getActorSection().isDisplayed();}, "Actor section is displayed!");
    Assertions.assertThrows(NoSuchElementException.class ,() ->
{findPage.getTitleSection().isDisplayed();}, "Title section is displayed!");
}
```

3.10. Search Filter Test

Test Name: searchWithTitleFilterTest

Description: Check if user is able to successfully search for a title with a filter.				
Pre-condition(s):				
Test Steps: 1. Enter title name in a search bar. 2. Click on a search filter button. 3. Click on a title filter button. 4. Click on search button. 5. Check if only title section is shown.	**Test Data:** Title name to be searched: House	**Expected Result:** User can successfully perform a search with a filter.	**Actual Result:** User can successfully perform a search with a filter.	**Status:** PASS
Notes:				

```
@Test
public void searchWithTitleFilterTest() {
   HomePage homePage = new HomePage(driver);
   FindPage findPage = new FindPage(driver);

   homePage.searchForTextTitleFilter("House");

   Assertions.assertTrue(findPage.getTitleSection().isDisplayed(), "Title
section is not displayed!");
   Assertions.assertThrows(NoSuchElementException.class, () ->
{findPage.getPeopleSection().isDisplayed();});
}
```

Test Name: searchWithoutFilterTest				
Description: Check if user is able to successfully search for a title without a filter with a different results.				
Pre-condition(s):				
Test Steps: 1. Enter title name in a search bar. 2. Click on search button. 3. Check if not only title section is shown.	**Test Data:** Title name to be searched: House	**Expected Result:** User can successfully perform a search without a filter with a different results than	**Actual Result:** User can successfully perform a search without a filter with a different results than when searching with a	**Status:** PASS

		when searching with a filter.	filter.	
Notes:				

```
@Test
public void searchWithoutFilterTest() {
   HomePage homePage = new HomePage(driver);
   FindPage findPage = new FindPage(driver);

   homePage.searchForText("House");

   Assertions.assertTrue(findPage.getTitleSection().isDisplayed(), "Title
section is not displayed!");
   Assertions.assertTrue(findPage.getPeopleSection().isDisplayed(), "People
section is not displayed!");
}
```

3.11. Rate Test

Test Name: guestUserRateTest				
Description: Check if a guest user is redirected to the sign in page when trying to rate a title.				
Pre-condition(s): User must not be signed in.				
Test Steps: 1. Enter title name in a search bar. 2. Click on search button. 3. Click on a first title that comes up. 4. Click on rate title button. 5. Click on the nitnth star. 6. Click on submit rating button. 7. Check if user is redirected to the sign in page.	**Test Data:** Title name to be searched: Reservoir Dogs	**Expected Result:** Guest user is redirected to the sign in page if he tries to rate a title.	**Actual Result:** Guest user is redirected to the sign in page if he tries to rate a title.	**Status:** PASS
Notes:				

```
@Test
public void guestUserRateTest() {
   FindPage findPage = new FindPage(driver);
   TitlePage titlePage = new TitlePage(driver);

   findPage.FindTitle("Reservoir Dogs");

   titlePage.rateTitle();

   WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
     wait.until((driver) -> ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete"));

   implicitWait(2000);
```

```
Assertions.assertTrue(driver.getCurrentUrl().contains("https://www.imdb.com/registration/signin"), "Guest user shouldn't be able to rate!");
}
```

Test Name: registeredUserRateTest				
Description: Check if user is able to rate a title successfully.				
Pre-condition(s): User must be signed in.				
Test Steps: 1. Sign in or load cookies. 2. Enter title name in a search bar. 3. Click on search button. 4. Click on a first title that comes up. 5. Click on rate title button. 6. Click on the nitnth star. 7. Click on submit rating button. 8. Check if rating is submitted.	**Test Data:** Title name to be searched: Reservoir Dogs	**Expected Result:** User is able to rate a title successfully.	**Actual Result:** User is able to rate a title successfully.	**Status:** PASS
Notes:				

```
@Test
public void registeredUserRateTest() {
   signInOrLoadCookies();
   FindPage findPage = new FindPage(driver);
   TitlePage titlePage = new TitlePage(driver);

   findPage.FindTitle("Reservoir Dogs");

   titlePage.rateTitle();

   implicitWait(2000);

   Assertions.assertTrue(titlePage.getTitleUserRating().contains("9\n" +
"/10"), "Can't rate the title!");
}
```

Test Name: removeRatingTest				
Description: Check if user is able to remove a rating successfully.				
Pre-condition(s): User must be signed in.				
Test Steps: 1. Sign in or load cookies. 2. Enter title name in a search bar. 3. Click on search button. 4. Click on a first title that comes up. 5. Click on rate title button. 6. Click on remove rating button. 7. Check if rating has been removed.	**Test Data:** Title name to be searched: Reservoir Dogs	**Expected Result:** User is able to remove a rating successfully.	**Actual Result:** User is able to remove a rating successfully.	**Status:** PASS
Notes:				

```
@Test
public void removeRatingTest() {
   signInOrLoadCookies();
   FindPage findPage = new FindPage(driver);
   TitlePage titlePage = new TitlePage(driver);

   findPage.FindTitle("Reservoir Dogs");

   titlePage.removeTitleRating();

   implicitWait(2000);

   Assertions.assertThrows(NoSuchElementException.class, () ->
{titlePage.getTitleUserRatingElement().isDisplayed();}, "Can't remove the
rating!");
}
```

3.12. Responsiveness Test

Test Name: testMobileResponsiveness				
Description: Check if website is responsive on a mobile device.				
Pre-condition(s):				
Test Steps: 1. Set window size to 375x800. 2. Check if nav bar IMDb logo is displayed properly. 3. Check if nav bar menu button is displayed properly. 4. Check if sign in button is displayed properly.	**Test Data:**	**Expected Result:** UI elements are displayed properly on a mobile device.	**Actual Result:** UI elements are displayed properly on a mobile device.	**Status:** PASS
Notes:				

```
@Test
public void testMobileResponsiveness() {
   HomePage homePage = new HomePage(driver);
   driver.manage().window().setSize(new Dimension(375, 800));

   Assertions.assertTrue(homePage.getNavBarLogo().isDisplayed(), "Logo is not
visible on mobile view");
   Assertions.assertTrue(homePage.getNavBarMenu().isDisplayed(), "Menu is not
visible on mobile view");
   Assertions.assertTrue(homePage.getNavBarSignInButton().isDisplayed(), "Sign
in button is not visible on mobile view");
}
```

Test Name: testMobileResponsiveness				
Description: Check if website is responsive on a tablet device.				
Pre-condition(s):				
Test Steps: 1. Set window size to 768x1024.	**Test Data:**	**Expected Result:** UI elements are	**Actual Result:** UI elements are displayed properly	**Status:** PASS

2. Check if nav bar IMDb logo is displayed properly. 3. Check if nav bar menu button is displayed properly. 4. Check if sign in button is displayed properly.		displayed properly on a tablet device.	on a tablet device.	
Notes:				

```
@Test
public void testTabletResponsiveness() {
   HomePage homePage = new HomePage(driver);
   driver.manage().window().setSize(new Dimension(768, 1024));

   Assertions.assertTrue(homePage.getNavBarLogo().isDisplayed(), "Logo is not
visible on tablet view");
   Assertions.assertTrue(homePage.getNavBarMenu().isDisplayed(), "Menu is not
visible on tablet view");
   Assertions.assertTrue(homePage.getNavBarSignInButton().isDisplayed(), "Sign
in button is not visible on tablet view");
}
```

3.13. Response Time Test

Test Name: testResponseTime				
Description: Check if website response time is acceptable.				
Pre-condition(s):				
Test Steps: 1. Send a GET request to the webpage and record a response time. 2. Check if response time is under 3 seconds.	**Test Data:**	**Expected Result:** Website response time is acceptable.	**Actual Result:** Website response time is acceptable.	**Status:** PASS
Notes:				

```
@Test
public void testResponseTime() {
    try {
        long responseTime = getResponseTime(Config.BASE_URL);
        assertTrue(responseTime < 3000, "Response time is too high!");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Test Name: testSlowNetworkResponseTime				
Description: Check if website response time is acceptable when accessed with slower network.				
Pre-condition(s):				
Test Steps: 1. Send a GET request to the webpage and record a response time. 2. Check if response time is under 3 seconds with simulated 1 second delay.	**Test Data:**	**Expected Result:** Website response time is acceptable.	**Actual Result:** Website response time is acceptable.	**Status:** PASS
Notes:				

```
@Test
public void testSlowNetworkResponseTime() {
    try {
        long responseTime = getResponseTime(Config.BASE_URL);
        responseTime += 1000; // Simulate 1-second delay due to slow network
connection
        assertTrue(responseTime < 3000, "Response time is too high!");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

3.14. HTTPS Test

Test Name: checkHTTPSTest				
Description: Check if website header containts STS protocol confirming HTTPS is actually enforced.				
Pre-condition(s):				
Test Steps: 1. Send a GET request to the webpage and record a response time. 2. Check if website header contains STS protocol.	**Test Data:**	**Expected Result:** Website header contains the STS protocol.	**Actual Result:** Website header contains STS protocol.	**Status:** PASS
Notes:				

```
@Test
public void checkHTTPSTest() {
   Assertions.assertTrue(checkHsts("https://www.imdb.com/"), "HSTS could not be
detected!");
}
```

3.15. XSS Test

Test Name: xssSearchBarInputTest				
Description: Website should handle XSS attacks.				
Pre-condition(s):				
Test Steps: 1. Input a XSS script in a search bar. 2. Click on a search button. 3. Check if bot check is displayed.	**Test Data:**	**Expected Result:** After user tries entering the XSS script, bot check is displayed.	**Actual Result:** After user tries entering XSS script, bot check is displayed.	**Status:** PASS
Notes:				

```
@Test
public void xssSearchBarInputTest() {
   HomePage homePage = new HomePage(driver);

   homePage.searchForText("<script>alert('XSS')</script>");

       Assertions.assertTrue(homePage.getCaptchaContainer().isDisplayed(),  "Bot
check not sent!");
}
```

Test Name: xssURLInputTest				
Description: Website should handle XSS attacks.				
Pre-condition(s):				
Test Steps: 1. Input a XSS script in URL. 2. Search for that URL. 3. Check if bot check is displayed.	**Test Data:**	**Expected Result:** After user tries entering the XSS script, bot check is displayed.	**Actual Result:** After user tries entering XSS script, bot check is displayed.	**Status:** PASS

Notes:

```
@Test
public void xssURLInputTest() {
   HomePage homePage = new HomePage(driver);


driver.navigate().to("https://www.imdb.com/find/?q=%3Cscript%3Ealert(%27XSS%27)
%3C/script%3E");

   Assertions.assertTrue(homePage.getCaptchaContainer().isDisplayed(), "Bot
check not sent!");
}
```

Test Name: xssSearchBarAlertLinkInputTest				
Description: Website should handle XSS attacks.				
Pre-condition(s):				
Test Steps: 1. Input a XSS script in a search bar. 2. Click on a search button. 3. Check if bot check is displayed.	**Test Data:**	**Expected Result:** After user tries entering the XSS script, bot check is displayed.	**Actual Result:** After user tries entering XSS script, bot check is displayed.	**Status:** PASS
Notes:				

```
@Test
public void xssSearchBarAlertLinkInputTest() {
   HomePage homePage = new HomePage(driver);

   homePage.searchForText("<a href='javascript:alert(\"XSS\")'>Click me</a>");

   Assertions.assertTrue(homePage.getCaptchaContainer().isDisplayed(), "Bot
check not sent!");
}
```

3.16. SQL Injection Test

Test Name: searchSQLInjectionTest				
Description: Website should handle SQL injection attacks.				
Pre-condition(s):				
Test Steps: 1. Input an SQL injection script in a search bar. 2. Click on a search button. 3. Check if bot check is displayed.	**Test Data:**	**Expected Result:** After user tries entering XSS script, bot check is displayed.	**Actual Result:** After user tries entering XSS script, bot check is displayed.	**Status:** PASS
Notes:				

```
@Test
public void searchSQLInjectionTest() {
   HomePage homePage = new HomePage(driver);

   String injectionString = "' OR '1'='1' --";

   homePage.searchForText(injectionString);

   Assertions.assertTrue(homePage.getCaptchaContainer().isDisplayed(), "Bot
check not sent!");
}
```

Test Name: signInSQLInjectionTest				
Description: Website should handle SQL injection attacks.				
Pre-condition(s):				
Test Steps: 1. Click on Navigation Bar Sign In button. 2. Click on Sign In with IMDb account button. 3. Type in SQL injection script in email and password fields.	**Test Data:**	**Expected Result:** After user tries entering SQL injection script, bot check is displayed.	**Actual Result:** After user tries entering SQL injection script, bot check is displayed.	**Status:** PASS

4. Click on submit button.				
Notes:				

```
@Test
public void signInSQLInjectionTest() {
   SignInPage signInPage = new SignInPage(driver);

   String injectionString = "admin' OR '1'='1' --";

   signInPage.signIn(injectionString, injectionString);

   Assertions.assertNotEquals("https://www.imdb.com/?ref_=login",
driver.getCurrentUrl(), "Sign in was successful!");
}
```

4. Conclusion

4.1. Testing Summary

Testing Tool	Total Tests	Passed Tests	Failed Tests
Selenium	50	50	0

4.2. Final Thoughts

IMDb as a biggest online database related to the media is obviously very well implemented and I haven't found any flaws in it's design. Functionalities seem more complex than average webpage possess and it might require much more thorough testing to find a design flaw.