Bilkent University

Department of Computer Engineering

CS 458 Software Verification and Validation

Project 1

Lecturer: Haluk Altunel

Prepared By

Yiğit Erkal-21601521

Gökberk Boz-21602558

Ahmet Ayberk Yılmaz-21702250

Ege Şahin-21702300

Contents

1. Netflix Login Page Codes **2**

2. Screenshots of the Application Interface **5**

3. UML Diagrams **7**

4. Examining Selenium **11**

5. Five Test Cases of Our Login Page **12**

6. Test Automation Code with Selenium-Webdriver **12**

7. Evaluation of our Automation Experience **15**

8. How Test Automation Contributes to Software Development Life Cycle in Terms of Velocity and Quality **16**

References **17**

1. Netflix Login Page Codes

```
const rememberedEmail = getEmailCookie();

function getEmailCookie() {
  let decodedCookies = decodeURIComponent(document.cookie);
  let cookiesArray = decodedCookies.split(";");
  for (let i = 0; i < cookiesArray.length; i++) {
    let cookie = cookiesArray[i];
    while (cookie.charAt(0) == " ") {
      cookie = cookie.substring(1);
    }
    if (cookie.indexOf("email=") == 0) {
      return cookie.substring("email=".length, cookie.length);
    }
  }
  return "";
}

emailInput.value = rememberedEmail;
```

Figure 1: Remember me code

This code piece gets the saved email from the cookie.

```
const inputOnBlur = () => {
  if (inputFocused.email) {
    if (!validateEmail(emailInput.value) && !validatePhone(emailInput.value)) {
      emailError.style.display = "block";
      emailInput.style.borderBottom = "2px solid #e87c03";
      inputFilled.email = false;
    } else {
      emailError.style.display = "none";
      emailInput.style.borderBottom = "none";
      inputFilled.email = true;
    }
  }
  if (inputFocused.password) {
    if (
      !(passwordInput.value.length >= 4 && passwordInput.value.length <= 60)
    ) {
      passwordError.style.display = "block";
      passwordInput.style.borderBottom = "2px solid #e87c03";
      inputFilled.password = false;
    } else {
      passwordError.style.display = "none";
      passwordInput.style.borderBottom = "none";
      inputFilled.password = true;
    }
  }
};
```

Figure 2: Input verification code

This code piece works when input is blurred then controls for the inputs if it is in the right format. If input is not right, then error comes under the input box.

```
document
  .querySelector("#signin-btn")
  .addEventListener("click", function (event) {
    event.preventDefault();
    wrongEmail.style.display = "none";
    wrongPassword.style.display = "none";
    wrongFacebook.style.display = "none";
    if (inputFilled.email && inputFilled.password) {
      if (rememberMe.checked) {
        document.cookie =
          "email=" +
          emailInput.value +
          "; expires=Thu, 1 Mar 2025 12:00:00 UTC";
      }
      for (let i = 0; i < users.length; i++) {
        if (users[i][0] == emailInput.value) {
          if (users[i][1] == passwordInput.value) {
            document.location.href = "successful.html";
            return;
          } else {
            wrongPassword.style.display = "block";
            return;
          }
        }
      }
      wrongEmail.style.display = "block";
    }
  });
```

Figure 3: Sign in code

This code piece runs when the sign-in button is clicked. If you remember me checkbox is ticked then it saves the username to cookies. Then it controls from the user array. If inputs are true then redirects to the successful page, if not then displays error.

```
window.fbAsyncInit = function() {
  FB.init({
    appId      : '321945806623532',
    cookie     : false,
    xfbml      : true,
    version    : 'v2.8'
  });

  FB.AppEvents.logPageView();

};

(function(d, s, id){
   var js, fjs = d.getElementsByTagName(s)[0];
   if (d.getElementById(id)) {return;}
   js = d.createElement(s); js.id = id;
   js.src = "https://connect.facebook.net/en_US/sdk.js";
   fjs.parentNode.insertBefore(js, fjs);
 }(document, 'script', 'facebook-jssdk'));
```

Figure 4: Facebook API connection code

This code piece runs when the html page first opens. It connects to Facebook API for Login with Facebook feature.

```
document
  .querySelector("#facebook-btn")
  .addEventListener("click", function (event) {
    event.preventDefault();
    wrongEmail.style.display = "none";
    wrongPassword.style.display = "none";
    wrongFacebook.style.display = "none";
    if (!facebookPopupOpened) {
      facebookPopupOpened = true;
      FB.login(function (response) {
        if (response.status === "connected") {
          document.location.href = "successful.html";
        } else {
          wrongFacebook.style.display = "block";
          facebookPopupOpened = false;
        }
      });
    }
  });
```

Figure 5: Facebook authentication code

This code piece runs when the Login with Facebook button is clicked. If the facebook login page is not opened it opens a webpage for login. After that, according to the result, either it redirects the user to successful page or gives an error.

```
document
.querySelector("#signout-btn")
.addEventListener("click", function (event) {
  event.preventDefault();
  FB.getLoginStatus(function (response) {
  if (response.status === "connected") {
    FB.logout(function (response) {
    // Person is now logged out
    });
  }
  });
  document.location.href = "index.html";
});
```

Figure 6: Sign out code

This code piece runs when the logout button is clicked in the successful page. If a user logged in with facebook it logged out from facebook. After that it redirects to the index page.

2. Screenshots of the Application Interface

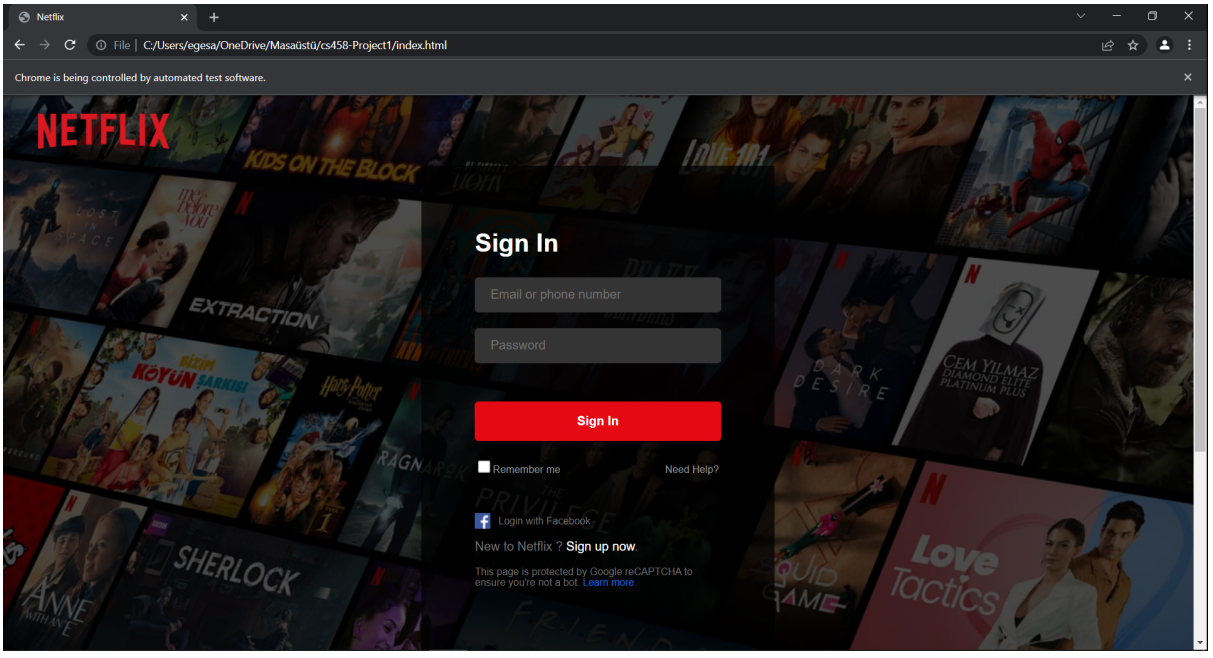

Figure 7: Login Page

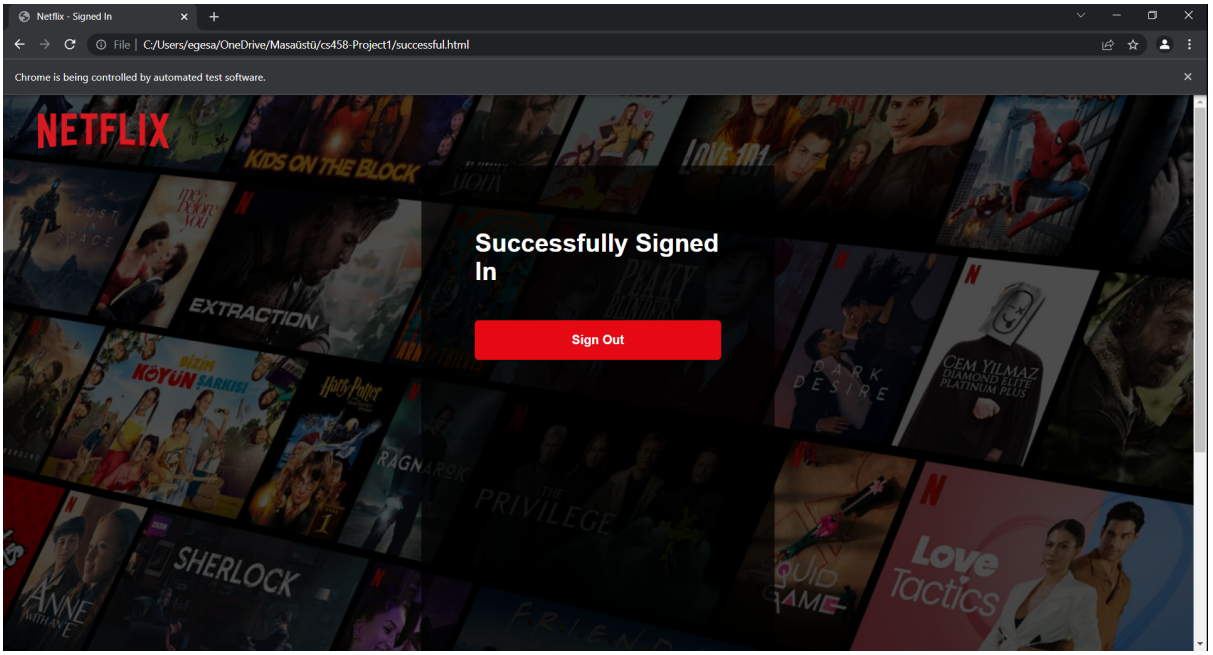

Figure 8 : The page shown after login is successful.

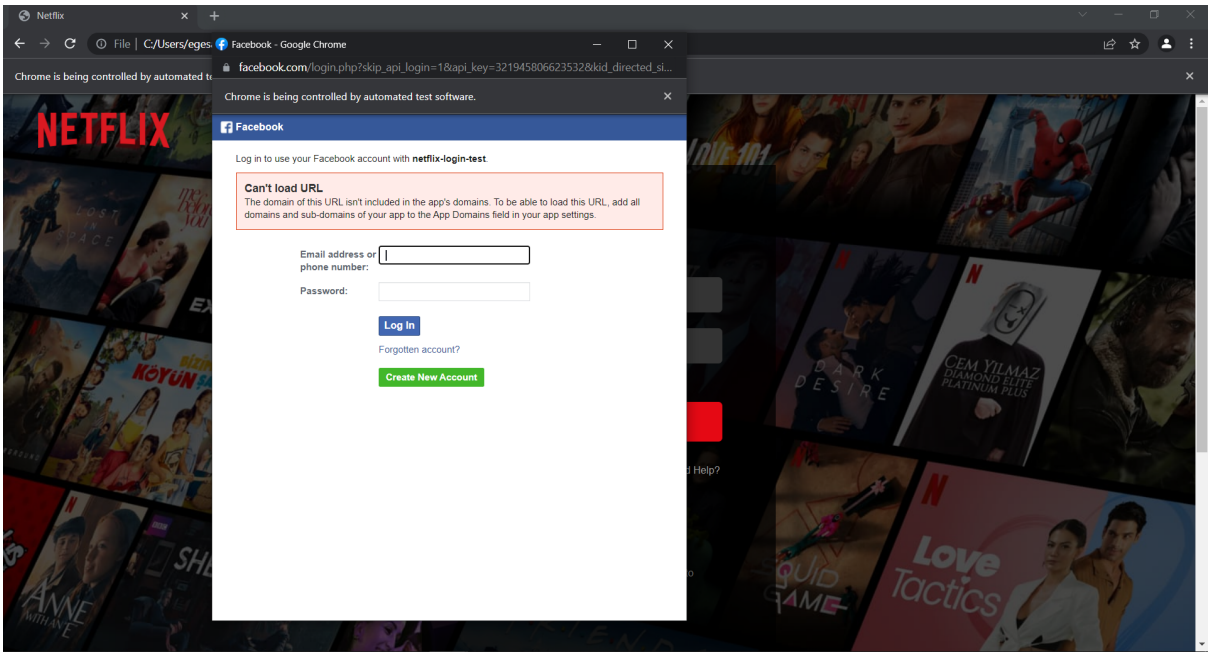

Figure 9: Facebook Popup (After login with facebook button is clicked).

3. UML Diagrams

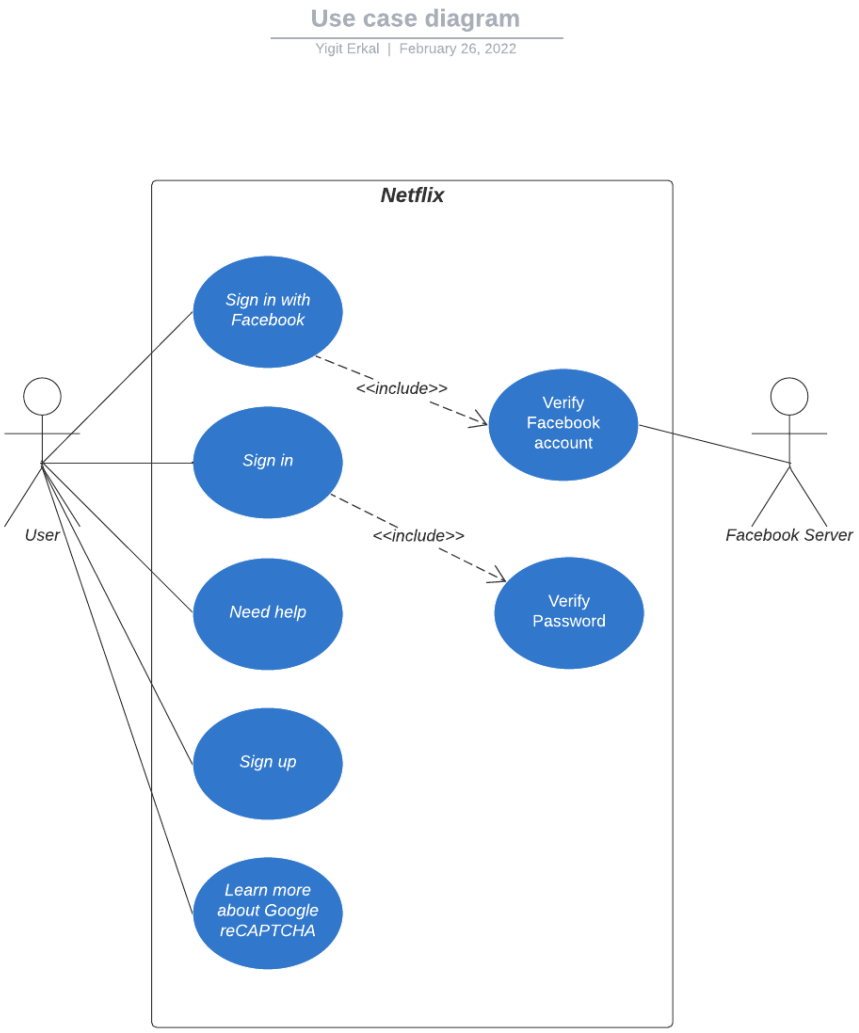

Figure 10: Use Case Diagram of Netflix Login

In the use-case diagram of the Netflix Login page, there are five functionalities. Users may sign in with the user account or sign in with Facebook. If a user signs in with the user account ,then, our login page verifies its username and password. If the user chooses to sign in with a Facebook account then a pop-up page occurs and the user enters the Facebook account details. Facebook server checks that the user is valid or not and according to the determination, the user signs in the Netflix page. In our Netflix sign in page, there are three more functionalities which are “Need Help” button, “Sign up button” and “Learn more button”.

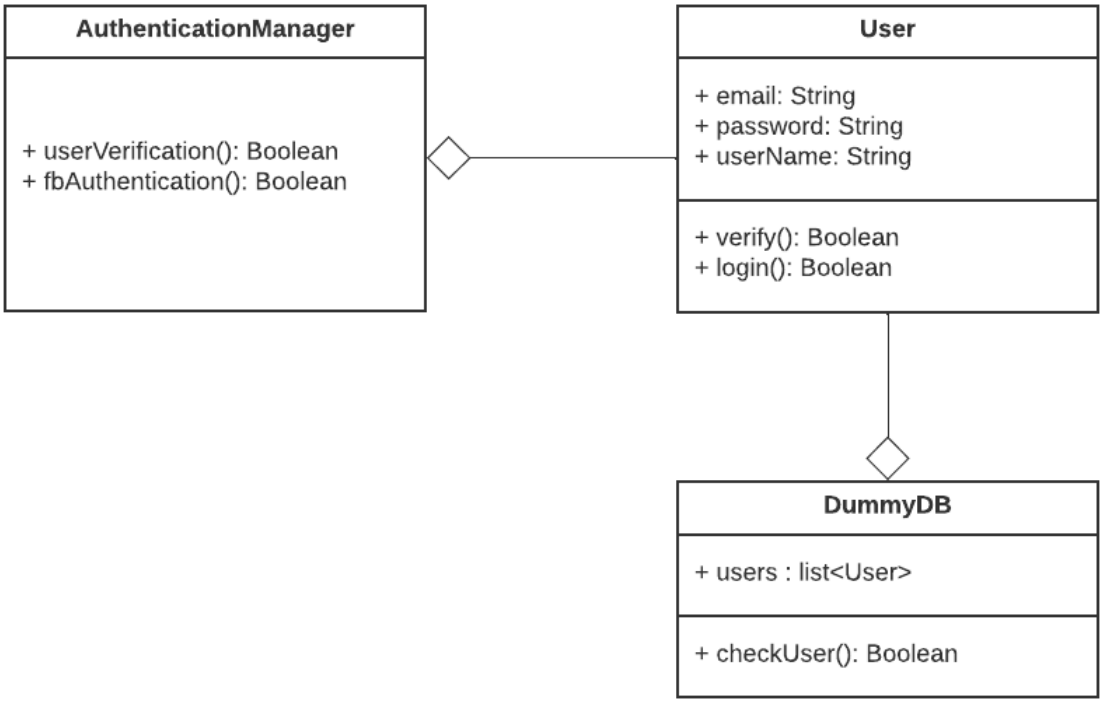

Figure 11: Class Diagram of Netflix Login

In the class diagram of the Netflix Sign in page, there are three classes. We have a user class which contains the data about the user. DummyDB class works like a database for our simple login page. We did not prefer to create a new database and we hold the user details in a list. Another class is AuthenticationManager that works for all authentication details of our page. userVerification works for our netflix login implementation and fbAuthentication works for facebook login.

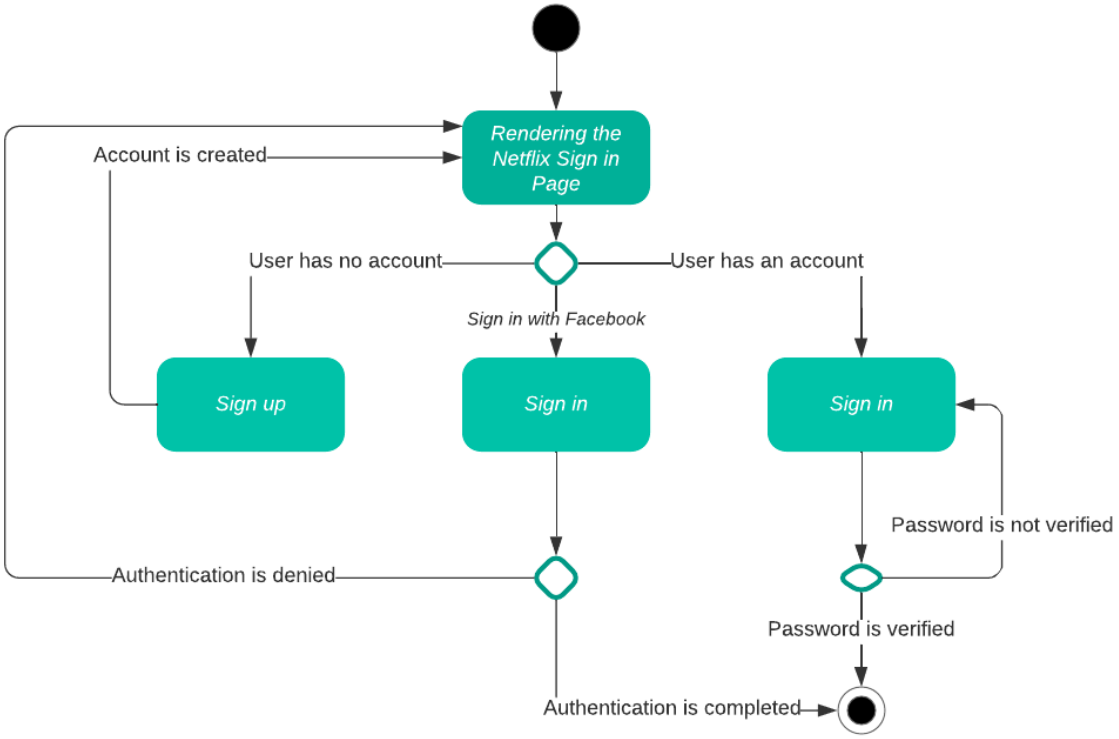

Figure 12: Activity Diagram of Netflix Login

In the activity diagram of Netflix login, initially, the user opens the our webpage. To sign in, users have two different options which are to sign in with Facebook or sign with an account. If the password and username verification is completed, the user signs in. If the authentication is denied, the sign in page renders again and waits until getting the correct user information. If a user does not have an account, there is a sign up button to create an account.

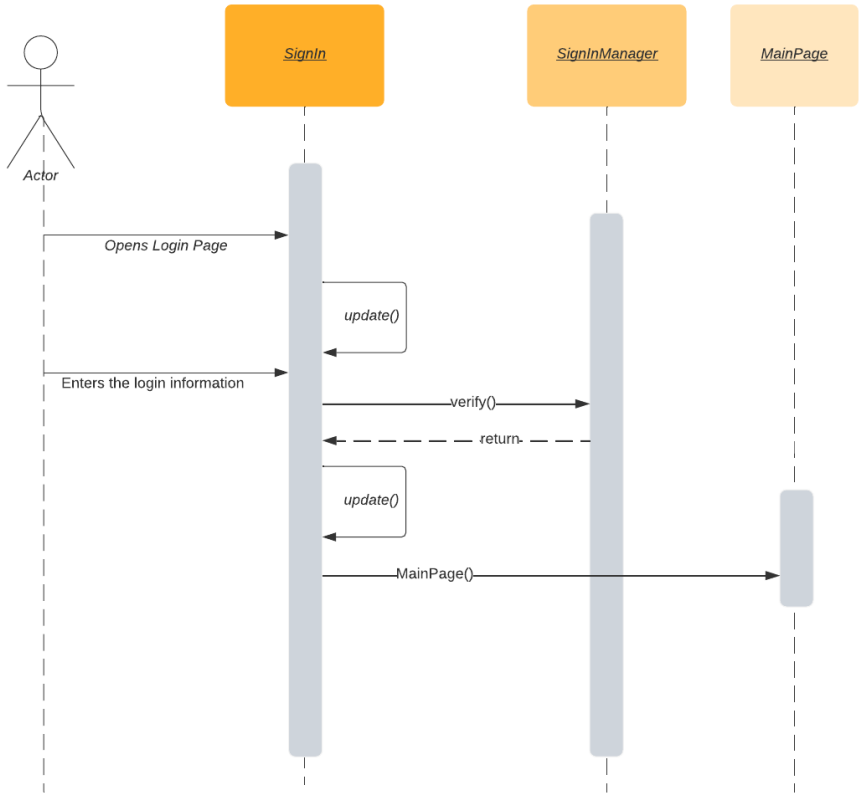

Figure 13: Sequence Diagram of Netflix Login

In the sequence diagram of Netflix login, the user opens the login page. Then, the user enters the user information which are email and password. Then, the verification process starts and if the user information details are correct, the user signs in and the website renders the main page.

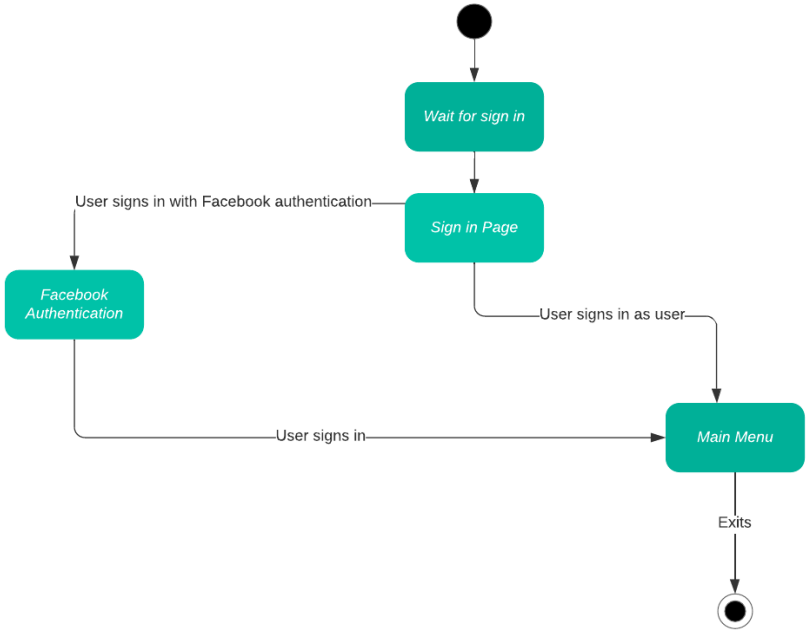

Figure 14: State Diagram of Netflix Login

The state diagram of the Netflix login has three states which are the sign in page, main menu and facebook authentication page. When the user opens the Netflix login page, the user has two options to login which are using Facebook or user login. After the login process is done, the user reaches the main menu page.

4. Examining Selenium

Open Source : It provides all the features completely free of cost. It can be downloaded directly [1].

Language Support : Multilingual support is one of the major benefits of Selenium WebDriver for automation testing. Python, PHP, Java, C#, Ruby, JavaScript etc [1].

Works Across Multiple OS : Selenium WebDriver supports multiple OS like Linux, UNIX, Mac as well as Windows [1].

Cross Browser Compatibility Testing : It supports all the major browsers so you could test on Chrome, Firefox, Safari, Opera, IE, Edge, Yandex and many more. When you are executing cross browser testing of a website, WebDriver provides you with an automated solution [1].

Cross-Device Testing : Automated test cases can now be written for testing on iPhones, Blackberry, and Android [1].

Easy to Implement : Since it is open source, it allows users to script their personalized extensions [1].

Mouse Cursor and Keyboard Simulation : Selenium Webdriver can mimic a real user scenario by handling mouse and keyboard events [1].

Server Starting Not Required : A major benefit of automation testing with Selenium WebDriver is that you don't need to start any server prior to testing [1].

Advanced Browser Interactions : With Selenium, it is possible to simulate advanced interactions like clicking the browser back and front buttons [1].

Parallel Test Execution : Using Selenium Grid execute tests can be executed in parallel, so that the test execution time can be reduced [2].

5. Five Test Cases of Our Login Page

Facebook Login : Trying directly login with Facebook pop-up. Both successful and unsuccessful logins are tried.

Netflix Login : Trying to login netflix. Both successful and unsuccessful logins are tried.

Remember User : Remembering the user information for login which saves email and password.

Different Browser : Trying other browsers for the same page.

Multiple Clicks on Login with Facebook Button : Login with facebook button is clicked consecutively to see if more than one pop-up will emerge.

6. Test Automation Code with Selenium-Webdriver

```
const { Builder, By, Key, until } = require("selenium-webdriver");
require("chromedriver");

var driver

async function succesfulLogin() {
    driver = await new Builder().forBrowser("chrome").build();

    await driver.get(__dirname + "/../index.html");

    await driver.findElement(By.name("email")).sendKeys("johndoe@mail.com");
    await driver.findElement(By.name("password")).sendKeys("123456789");
    await driver.findElement(By.id("signin-btn")).click();

    await driver.sleep(5000);
    await driver.quit();
}
```

Figure 15: Successful login test function code

```
async function unsuccesfulLogin() {
    driver = await new Builder().forBrowser("chrome").build();

    await driver.get(__dirname + "/../index.html");

    await driver.findElement(By.name("email")).sendKeys("johndoemail.com");
    await driver.findElement(By.name("password")).sendKeys("123456789");
    await driver.findElement(By.id("signin-btn")).click();

    await driver.sleep(5000);
    await driver.quit();
}
```

Figure 16: Unsuccessful login test function code

The two functions above, test the netflix sign in page. In the first function a registered user simulated and she logged in. This leads her to the Netflix homepage. In the second function an unregistered user is simulated to login to Netflix and this results in an error message in the login page. These two functions are written as two sides of a single test case. Also these two functions use chrome as a browser.

```
async function succesfulLoginFirefox() {
    driver = await new Builder().forBrowser("firefox").build();

    await driver.get(__dirname + "/../index.html");
    await driver.findElement(By.name("email")).sendKeys("johndoe@mail.com");
    await driver.findElement(By.name("password")).sendKeys("123456789");
    await driver.findElement(By.id("signin-btn")).click();

    await driver.sleep(5000);
    await driver.quit();
}
```

Figure 17: Different browser which is Firefox login test code

The succesfulLoginFirefox() function tries to open the netflix sign in page with a different browser (in this case firefox). After opening the page with the browser the function simulates the login process of a registered user. So this function actually tests the Cross Browser Compatibility of Selenium.

```
async function facebookLogin() {
    driver = await new Builder().forBrowser("chrome").build();

    await driver.get(__dirname + "/../index.html");
    await driver.findElement(By.id("facebook-btn")).click();

    var parent = await driver.getWindowHandle();
    var windows = await driver.getAllWindowHandles();

    await driver.switchTo().window(windows[1]);

    await driver.findElement(By.id("email")).sendKeys("helena438@gmail.com");
    await driver.findElement(By.id("pass")).sendKeys("nekroz453");
    await driver.findElement(By.id("loginbutton")).click();

    await driver.sleep(8000);

    driver.close();
    driver.switchTo().window(parent);
    await driver.sleep(2000);
    await driver.quit();
}
```

Figure 18: Login with Facebook authentication test code

```
async function unsuccesfulfacebookLogin() {
    driver = await new Builder().forBrowser("chrome").build();

    await driver.get(__dirname + "/../index.html");
    await driver.findElement(By.id("facebook-btn")).click();

    var parent = await driver.getWindowHandle();
    var windows = await driver.getAllWindowHandles();

    await driver.switchTo().window(windows[1]);

    await driver.findElement(By.id("email")).sendKeys("johndoe@mail.com");
    await driver.findElement(By.id("pass")).sendKeys("123456789");
    await driver.findElement(By.id("loginbutton")).click();

    await driver.sleep(15000);

    driver.close();
    driver.switchTo().window(parent);
    await driver.sleep(2000);
    await driver.quit();
}
```

Figure 19: Unsuccessful login with Facebook authentication test

These two functions simulate the login with facebook functionality of the web page. In the first function chrome browser opens the sign in page and then login with facebook button is clicked by selenium. After that a new page is opened. Driver is switched between windows to manipulate the second (newly opened) screen. After that driver simulates the login of a registered facebook user and facebook accepts that.
The second function is very similar to the first one. The only difference is that an unregistered facebook user is simulated to enter with facebook and facebook rejects this entrance.
These two functions are written as two sides of a single test case.

```
async function rememberMe() {
    driver = await new Builder().forBrowser("chrome").build();

    await driver.get(__dirname + "/../index.html");

    await driver.findElement(By.name("email")).sendKeys("johndoe@mail.com");
    await driver.findElement(By.name("password")).sendKeys("123456789");

    await driver.findElement(By.id("rememberMe")).click();

    await driver.sleep(1000);

    await driver.findElement(By.id("signin-btn")).click();

    await driver.sleep(1000);

    await driver.findElement(By.id("signout-btn")).click();

    await driver.sleep(2000);
    await driver.quit();
}
```

Figure 20: Remember me functionality test code

The rememberMe() function simulates a click on the remember me checkbox when a user tries to login. This function simulates filling the email and password text fields and after that clicks on the remember me checkbox. After that function clicks on the sign in button and the login page changes to another

page which indicates that the login is successful. The function continues and click sign out button. Then the screen changes back to the login page and the email and password text fields are remembered.

```
async function multipleFacebookButtonClick() {
    driver = await new Builder().forBrowser("chrome").build();

    await driver.get(__dirname + "/../index.html");
    await driver.findElement(By.id("facebook-btn")).click();
    await driver.findElement(By.id("facebook-btn")).click();
    await driver.findElement(By.id("facebook-btn")).click();
    await driver.findElement(By.id("facebook-btn")).click();
}
```

Figure 21: Multiple time clicking Facebook authentication test code

The multipleFacebookButtonClick() method is testing the behavior ofthe login with facebook button. In this test, four consecutive clicks on login with a facebook button are simulated. The webpage responds as just opening one new tab for facebook as expected.

```
succesfulLogin();
rememberMe();
facebookLogin();
succesfulLoginFirefox();
multipleFacebookButtonClick()
unsuccesfulLogin();
unsuccesfulfacebookLogin();
```

Figure 22: Functions for testing

These are the function calls of the explained functions at the end of the test.js file.

7. Evaluation of our Automation Experience

In the beginning, Selenium was a new tool for all of us. Therefore, we started to do the research about the automation tool which is Selenium. Then, we started to try Selenium with simple scripts such as clicking the button, opening the webpage and switching the frame. When we all got an experience about Selenium, we determined the test cases. Our chosen test cases were giving the wrong/correct ID to sign in, trying to connect with Facebook authentication and giving the wrong/correct ID, checking the need help button, checking the UI of our page in different browsers which are Google Chrome and Mozilla Firefox and as a last using the remember me button.

Initially, we decided to use JavaScript to prepare our automation scripts and included the required web drivers such as chromedriver for Google Chrome and geckodriver for Mozilla Firefox. Writing the automation scripts was easier than our expectations. For instance, showing which button will be clicked or which input will be filled only requires the id of that element. However, some of the id's were hard to find such as in Facebook authentication page. Because, Facebook was using the dynamic id for the login button. Each time we tried to automate, the id of the button was changing. To solve that problem, we searched for unique information for the button in the higher level of the HTML. Also, we learned how to inspect the web pages deeply when we were facing this problem. Another hard thing for

us was switching frames when we were preparing Facebook authentication for automation. However, with the help of our research, we finished that case completely.

Consequently, we can say that automation was a completely unseen feature for us and after this project we all have some experience about Selenium. We also liked the things that we can do after learning this automation tool. For instance, we can automate some sign in pages that we frequently use or when we need a dummy email, we can prepare the script for getting a new Gmail account much faster. Shortly, our automation experience was funny and informative because we all had fun while we were preparing the scripts. Also, it was informative because we learned Selenium web drivers and in addition to that we learned how to inspect the web pages and how to find unique identifiers to the elements.

8. How Test Automation Contributes to Software Development Life Cycle in Terms of Velocity and Quality

Test Automation was a new concept for all of us and actually seemed unnecessary when it was introduced. Since the projects and homeworks are not too complicated for exerting effort to learn and implement automation tests. Testing the parts manually was the preferred method. However, the process and the outcomes were contrary to what we expected.

The first benefit of the test automation was forcing us to test the program with many different scenarios. In manual testing, generally a feature is tested once or twice. For example, “remember me” button is clicked and close or refresh the application, execute it again and see if it is there or not. But in automation testing there were many scenarios to try. Such as, overwriting information, trying from different browsers, trying to login with wrong information so that “remember me” remembers the wrong information or not remembering at all, etc. With these different scenarios we saw the incomplete and missing parts of the application that may decrease performance. So in terms of application quality, the automation test system increased the application’s functionality and performance.

Another benefit of automation testing is that it speeds up the whole process. Each manual testing scenario requires an execution time, data clear, applying test data to the program, etc. However, the automation test does many of the pre-requests itself which saves time and work.

Finally, the automation testing increased the application quality and development process even in a small scale project. And its efficiency definitely would be increased at a business level.

References

[1] G. C. Reddy. “Software Testing”. Online:
https://www.gcreddy.com/2021/06/advantages-and-disadvantages-of-selenium.html
Last Update: June 30, 2021.

[2] Arnab Roy Chowdhury. “Why Selenium WebDriver Should Be Your First Choice for Automation Testing” Online:
https://www.lambdatest.com/blog/13-reasons-why-selenium-webdriver-should-be-your-first-choice-for-automation-testing/
Last Update: January 24, 2019