Bilkent University

Department of Computer Engineering

Verification & Validation

Netflix Login

Project 1

Maryam Shahid	21801344
Hammad Khan Musakhel	21801175
Murat Sevinç	21702603

March 1, 2022

Table of Contents

1. UML Models

1.1. Use-Case Model

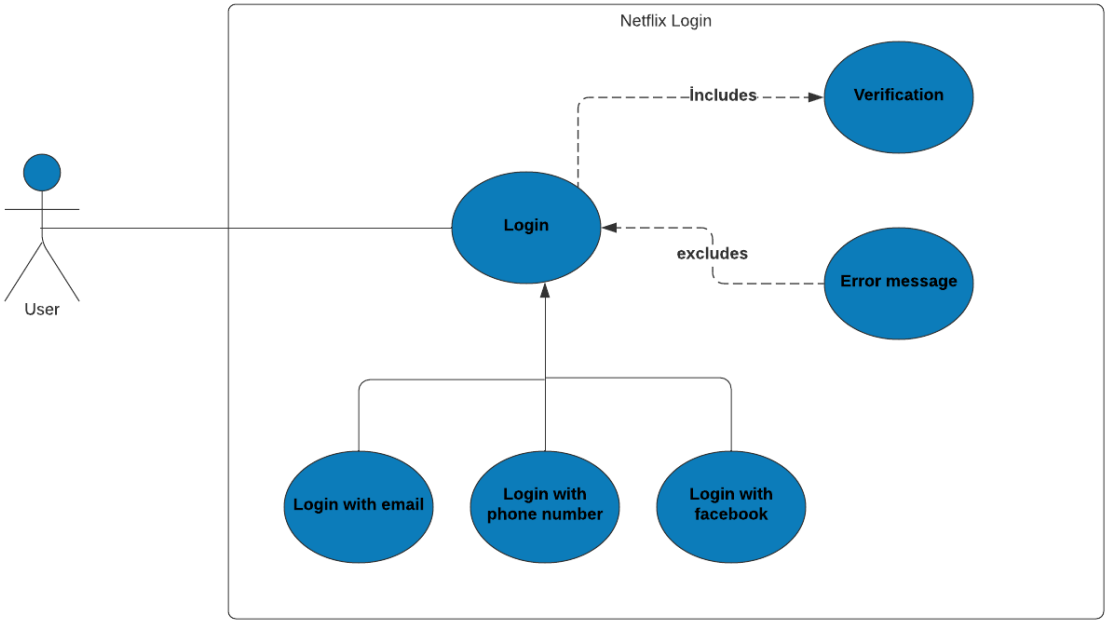


Figure 1: Use Case Diagram for Login Functionalities

1.1.1. Login

- **Participating actor:** User
- **Entry Condition:**
  - User enters login credentials with regards to email
  - User enters login credentials with regards to phone number
  - User enters login credentials with regards to facebook account
- **Exit condition:**
  - Login credentials are correct
- **Flow of Events:**
  - User enters credentials
  - Server checks credentials against database
  - If correct, user logs in

- ➢ If incorrect, error message is displayed

❖ **Special Requirements:**

- ➢ **None**

1.2. Activity Model

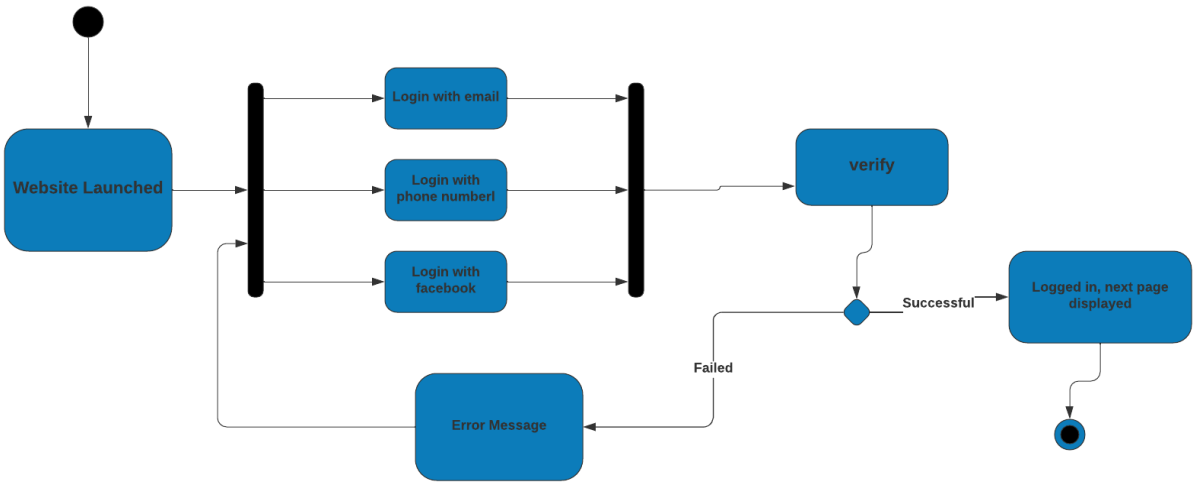


Figure 2: Activity Model For Login Page

Once the website is launched, the login page prompts the user to enter login credentials, be it with email, phone number, or facebook account. If the user enters inappropriate credentials with regards to any of the login methods, an error message is displayed and he/she can re-enter. Additionally, if the user provides authentic credentials which are tested against the database, the user is logged into the next page.

1.3. State Model

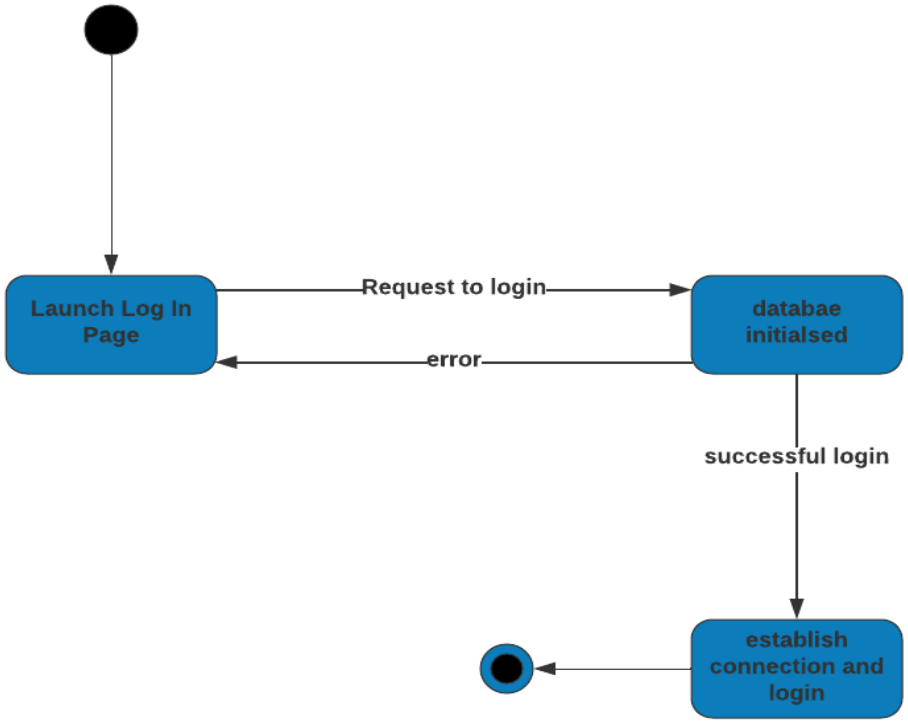


Figure 3: State Model for Login Page

Once the website is launched, the system takes the form of a login page. If the user enters inappropriate credentials with regards to any of the login methods, the state is not changed. Additionally, if the user provides authentic credentials which are tested against the database, the user is logged into the next state of the logged in page.

1.4. Class Model

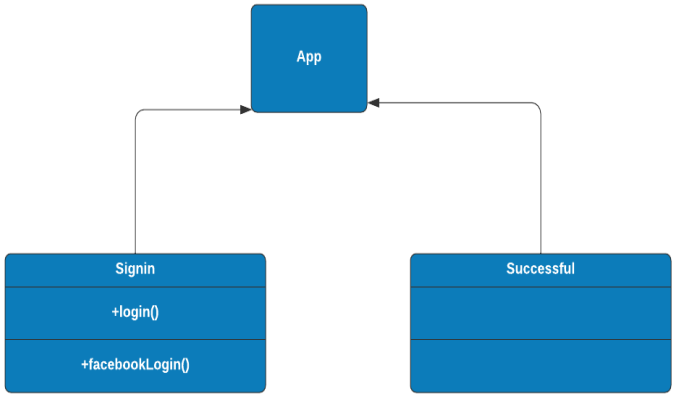


Figure 4: Class Model for Login Page

The web application has been developed on Vue.Js, thus, classes have not been used. However, in the script part, three components have been used and the signin component has the methods of Login() and facebookLogin().

1.5. Sequence Model

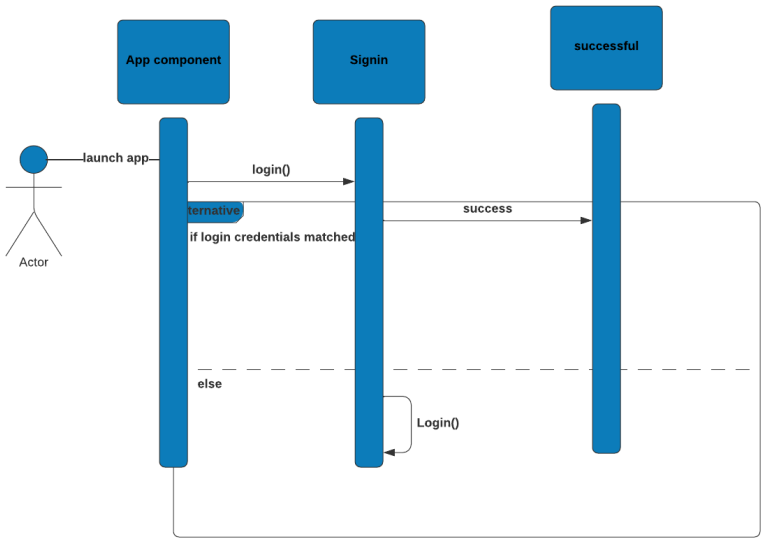


Figure 5: Sequence Model Successful Login

2. Implementation

2.1. GUI

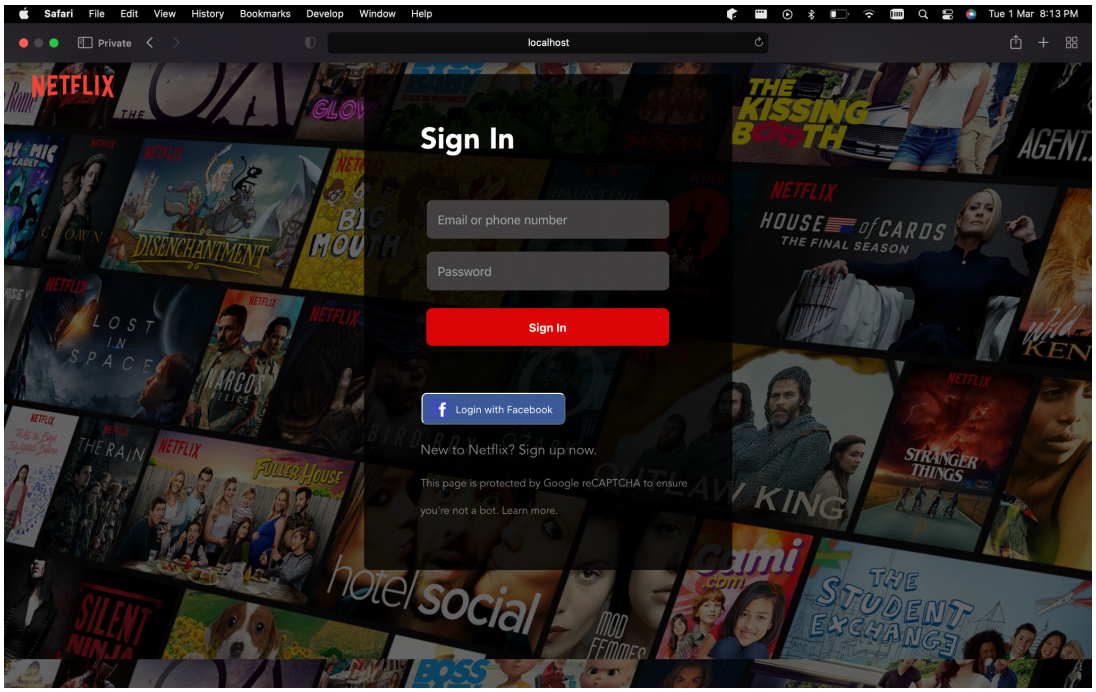


Figure 6: Netflix Sign-in Page

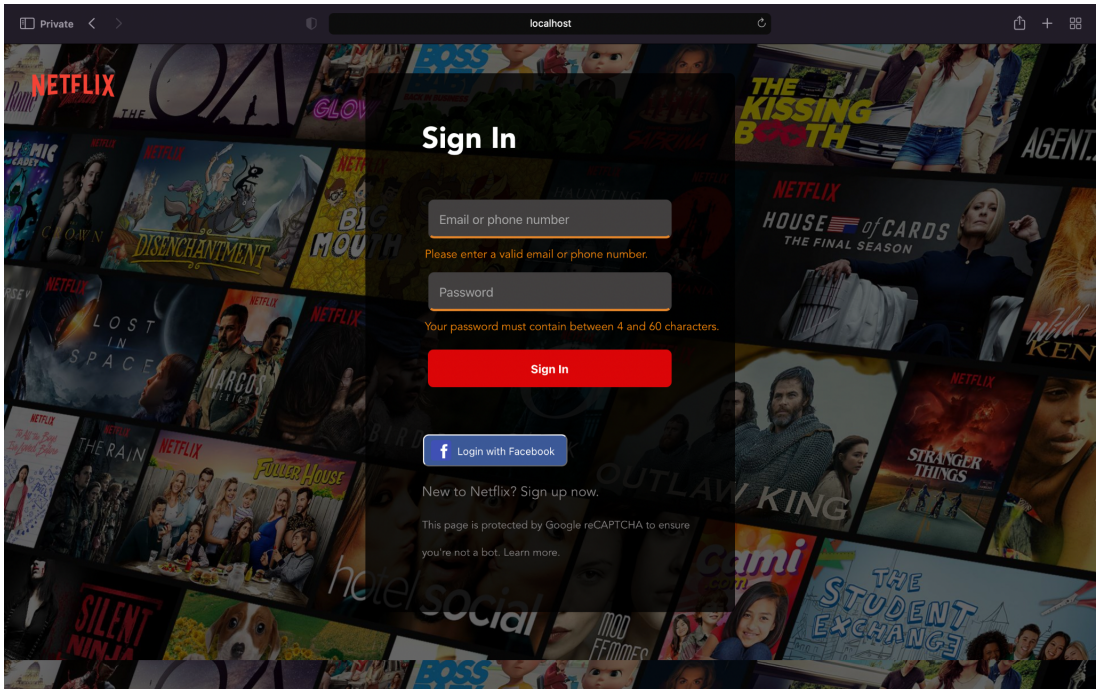
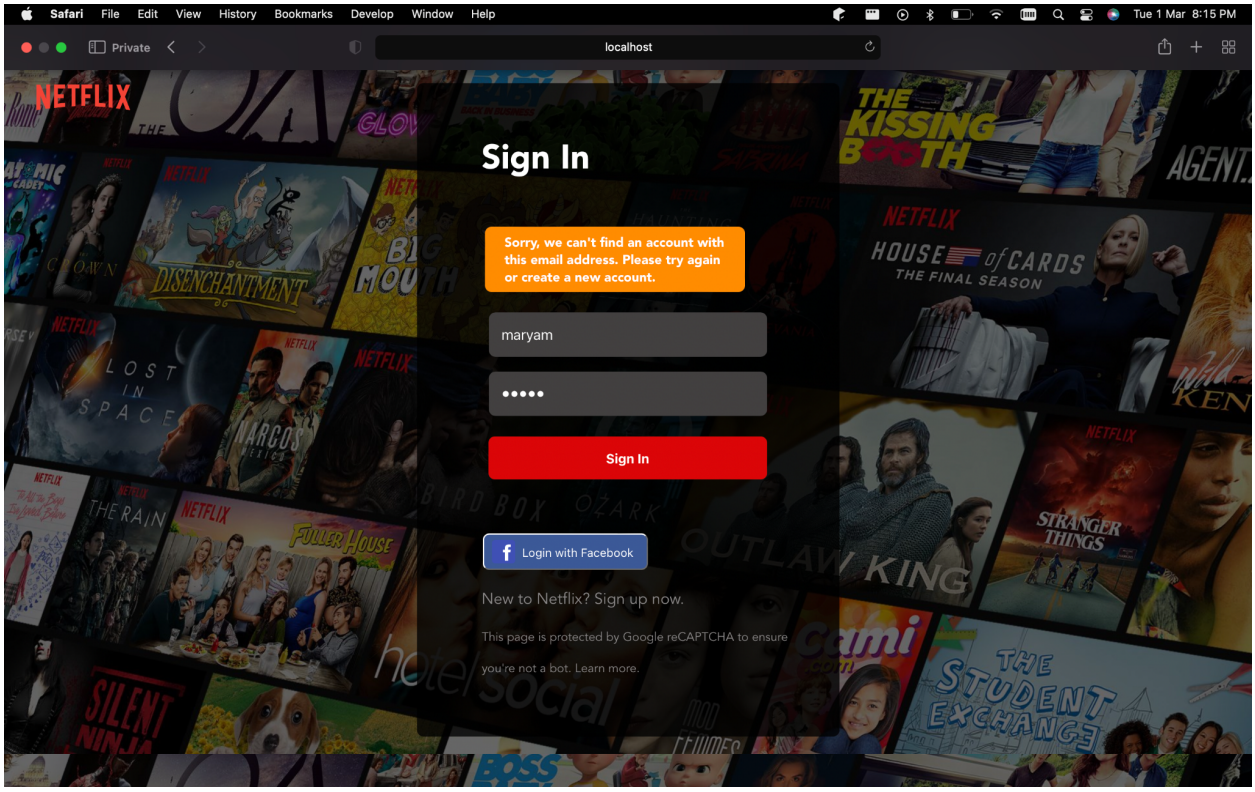


Figure 7: Sign-in with empty email and password


Safari File Edit View History Bookmarks Develop Window Help
Tue 1 Mar 8:15 PM
Private
localhost
NETFLIX
Sign In
Sorry, we can't find an account with this email address. Please try again or create a new account.
maryam
Sign In
Login with Facebook
New to Netflix? Sign up now.
This page is protected by Google reCAPTCHA to ensure you're not a bot. Learn more.

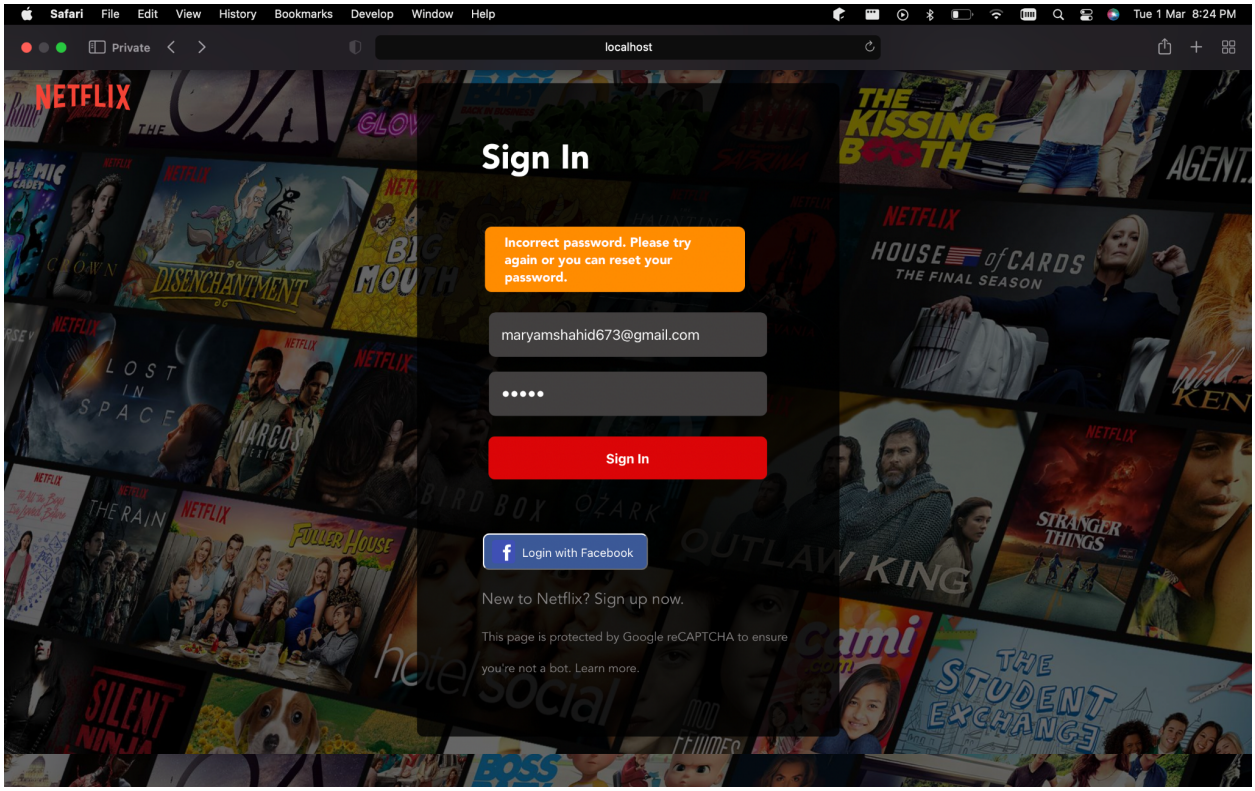

Safari File Edit View History Bookmarks Develop Window Help
Tue 1 Mar 8:24 PM
Private
localhost
NETFLIX
Sign In
Incorrect password. Please try again or you can reset your password.
maryamshahid673@gmail.com
Sign In
Login with Facebook
New to Netflix? Sign up now.
This page is protected by Google reCAPTCHA to ensure you're not a bot. Learn more.

Figure 8: Sign in with incorrect username and password

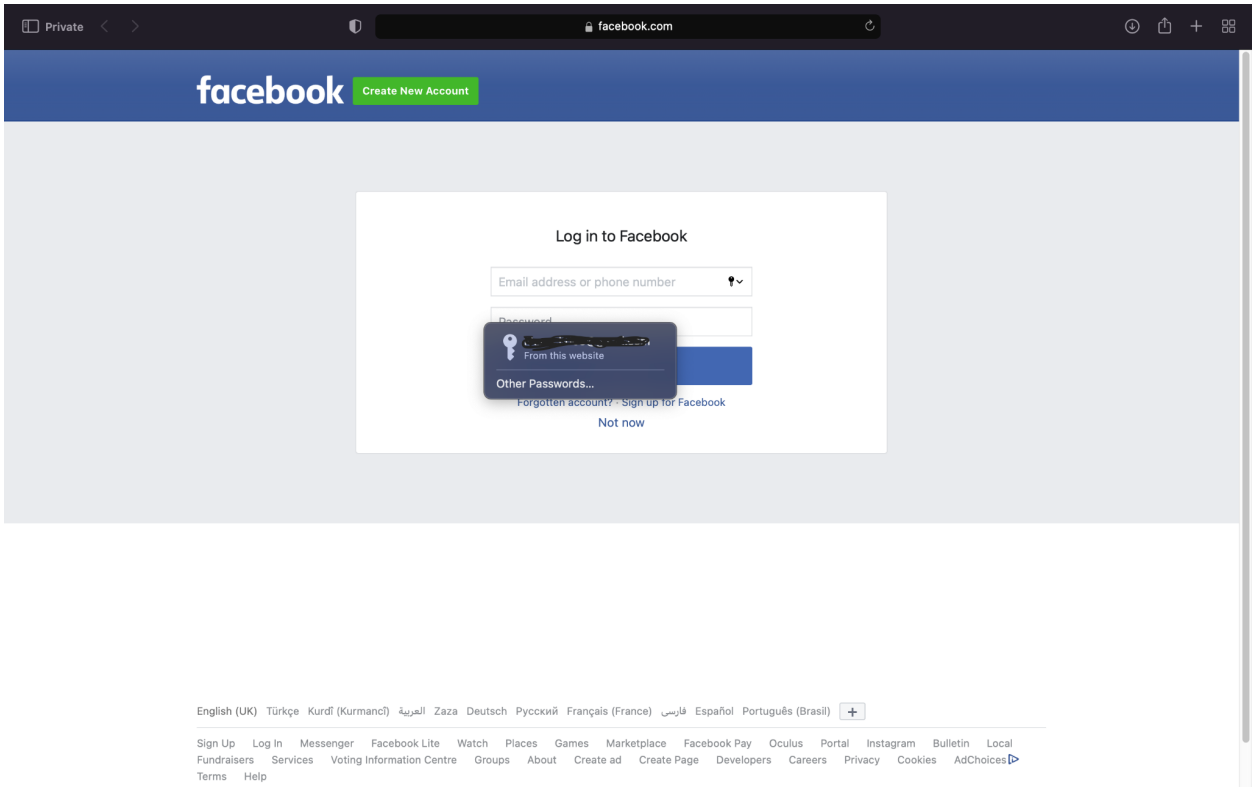


Figure 9: Sign in with facebook page

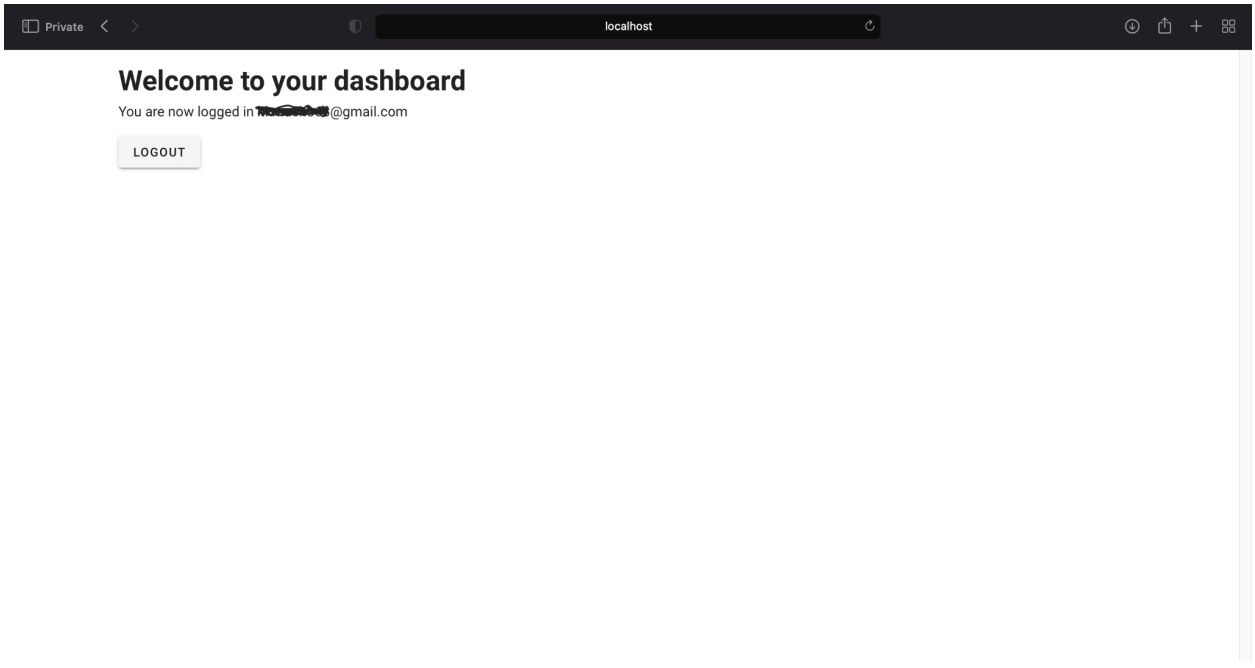


Figure 10: After a successful login

2.2. Implementation

The project was implemented in VueJs for frontend and Firebase for backend and data management. For the front-end, three pages were created: Netflix Sign-in, Facebook Sign-in and Success Dashboard. The full implementation of the project can be seen at: https://github.com/maryamShahid/cs458-project-1. [1]

The html part of the Netflix sign-in page can be seen in the figure below.

```
<template>
  <div class="card">
    <img src="./../assets/netflix-icon.svg" alt="" style="float: left; margin-left: 50px">
    <div class="outer">
      <div class="middle">
        <div class="inner">
          <div class="login-page__start_form">
            <div class="login-page__start_form_input">
              <p>Sign In</p>
              <div v-if="errorMessage" class="error"> {{ errorMessage }}</div>
              <div :class="{'login-input-error': usernameError}">
                <input v-model="username" @click="usernameError = false" placeholder="Email or phone number" @keyup.enter="register">
              </div>
              <div v-if="usernameError" class="error-text">Please enter a valid email or phone number.</div>
              <div class="login-page__start_form_input">
                <div style="position: relative">
                  <div :class="{'login-input-error': passwordError}" style="margin-top: 20px">
                    <input v-model="password" :type="hidden ? 'password' : 'text'" class="loginInput" @click="passwordError = false" placeholder="Password" @key
                  </div>
                </div>
                <div v-if="passwordError" class="error-text">Your password must contain between 4 and 60 characters.</div>
              </div>
              <button class="loginBtn" @click="register">Sign In</button>
              <div class="description" style="...">
                <button class="fb-button" @click="login">
                  <img alt="" src="./../assets/fb-icon.svg" style="margin-bottom: -16px">
                  Login with Facebook
                </button>
              </div>
              <div class="description">New to Netflix? Sign up now.</div>
              <div class="description-2">This page is protected by Google reCAPTCHA to ensure you're not a bot. Learn more.</div>
            </div>
          </div>
        </div>
      </div>
```

Figure 11: Netflix sign-in page HTML

As seen above, a sign-in form was created consisting of two inputs. The inputs were binded with username and password using v-model. Two buttons were added for log-in and log-in with facebook using the button component of VueJs.

Error states for empty fields and incorrect fields were added using the v-if component of VueJs and their respective classes were created.

For routing between pages, vue-routers were used. The router information can be seen in the main.js file shown in figure 12. It also shows the firebase configuration. This configuration allows us to connect to the application in firebase and easily use the authentication application of firebase to verify login with facebook, email, and others.

```
import { createApp } from 'vue'
import {createRouter, createWebHistory} from "vue-router"
import App from './App.vue'
import Home from '@/views/SignIn'
import SuccessPage from '@/views/SuccessPage'
import FacebookPage from '@/views/FacebookPage'
import firebase from '/firebase'

const router = createRouter( options: {
    history: createWebHistory(),
    routes: [
        {path: '/', name:'HomePage', component: Home},
        {path: '/success', name:'SuccessPage', component: SuccessPage},
        {path: '/facebook', name:'FacebookPage', component: FacebookPage}
    ]
})

const firebaseConfig = {
    apiKey: "AIzaSyCEDDzq-sDWVvR6bE9N-FkI9yVfrmSNjYs",
    authDomain: "netflix-f457c.firebaseapp.com",
    projectId: "netflix-f457c",
    storageBucket: "netflix-f457c.appspot.com",
    messagingSenderId: "545776945836",
    appId: "1:545776945836:web:3568c4630a2c850c5e6d09",
    measurementId: "G-WVD74GGZ09"
};

firebase.initializeApp(firebaseConfig);

createApp(App)
    .use(router)
    .mount( rootContainer: '#app')
```

Figure 12: Main.js file

```
facebookLogin(){
  let that = this
  const provider = new $nuxt.$fireModule.auth.FacebookAuthProvider()
  this.$fire.auth.signInWithPopup(provider)
    .catch(function (error){
      that.snackbarText = error.message
      that.snackbar = true
    }).then((user) => {
    //we are signed in
    $nuxt.$router.push('/')
  })
},
```

```
  '@nuxtjs/axios',
  // https://go.nuxtjs.dev/pwa
  '@nuxtjs/pwa',

    [
      '@nuxtjs/firebase',
      {
        config: {
          apiKey: "AIzaSyB21rFOn57_EZhuonQokv4vbP-_z0y5QrQ",
          authDomain: "authentication-b21be.firebaseapp.com",
          projectId: "authentication-b21be",
          storageBucket: "authentication-b21be.appspot.com",
          messagingSenderId: "685863118652",
          appId: "1:685863118652:web:06b6d297fa591853bd2039",
          measurementId: "G-0LN0ZW301C"
        },
        services: {
          auth: {
            persistence: 'local', // default
            initialize: {
              onAuthStateChangedAction: 'onAuthStateChangedAction',
              subscribeManually: false
            },
            ssr: false,
          }
```

Figure 13, 14: Code for facebook login method and nuxt.config.js

The facebook login needed to be implemented using an API. There are multiple ways to approach this challenge and we resorted to using the firebase's authentication feature for this. Firebase and Vue.Js have the compatibility. The Nuxt.js framework of Vue.Js applications has an easy going compatibility with firebase. One can easily install the nuxt-firebase package, add the firebase configurations to the nuxt.config.js. When we first developed the application, since we did not have to use a backend API for email and phone number, we continued with using the simple way of implementing it all within our front-end. However, upon reaching the facebook login, we faced

difficulty in incorporating the facebook API by simply using Vue.Js. Therefore, we incorporated Nuxt.Js and Firebase into the project.

3. Examining Selenium

Selenium is a software to help test automation generally used for web applications. It can automate browsers just like a user.

Capabilities

3.1. Starting A Webdriver

Selenium can initialize & automate different kinds of browsers (Chrome, Edge, Firefox, etc.)

```
WebDriver webDriver = new FirefoxDriver();
```

3.2. Getting a URL

With an initialized webdriver, selenium can go to any URL.

```
driver.get(netflixURL);
```

3.3. Finding an Element

After selenium goes to a URL, it can find web elements that are present. There are many ways to find a web element for selenium e.g. By XPath, className, id or CSS Selector.

```
driver.findElement(By.xpath(loginButtonXPath))
```

3.4. Clicking On An Element

Selenium can click web elements with the click() method.

```
driver.findElement(By.xpath(loginButtonXPath)).click();
```

3.5. Taking A Screenshot

While testing, screenshots might be quite important. Thus, selenium has this capability as well.

```
driver.getScreenshotAs(OutputType.BASE64);
```

3.6. Adding A Cookie

Cookies are now used widely in web development. With Selenium, artificial & unnatural cookies can be added.

```
driver.manage().addCookie(new Cookie( name: "key",  value: "value"));
```

3.7. Getting the Text of an Element

This practice is widely used in testing to compare results. Selenium can extract the text information of an element.

```
assertThat(errorMessage1.getText()).isEqualTo("Plea
```

3.8. Waiting for a Condition

Generally web applications have visual effects, this makes some web elements appear a bit later. Waiting strategies are important to catch those elements.

```
WebElement element = wait.until(
        ExpectedConditions.visibilityOfElementLocated(By.id("someid")));
```

3.9. Sending Keys

Almost in every test, some input areas are filled. Selenium can mimic the keyboard and send keys to that input.

```
driver.findElement(By.xpath(idInputXPath)).sendKeys(validPhoneNumber);
driver.findElement(By.xpath(passwordInputXPath)).sendKeys(validPassword);
```

3.10. Switching Between Windows

To test a pop-up window, this functionality is needed. Selenium can switch between tabs and windows.

```
driver.switchTo().newWindow(WindowType.TAB);
driver.switchTo().newWindow(WindowType.WINDOW);
```

4. Testing

4.1. Test Case 1: Simultaneous Connections

The robustness of the application is tested with numerous parallel connections to the application. This case basically tests whether the application works and shows the content to 30 simultaneous users. (Note that hardware is limited, with more advanced hardware, the number 30 should be increased to 30 thousand to test the actual robustness).

```
@org.junit.Test
public void applicationResponsesMultipleUsersSimultaneously() {
    IntStream.range(0, 30).parallel().forEach(el -> {
        ChromeDriver driver = createInvisibleChromeDriver();
        driver.get(netflixURL);

        WebElement loginButton = driver.findElement(By.xpath(loginButtonXPath));

        assertThat(loginButton.isDisplayed()).isTrue();
    });
}
```

4.2. Test Case 2: Invalid Inputs

The aim of this test case is to validate the application with the SRS (Software REquirement Specification). Basically, wrong inputs are provided and an error message is expected.

```
@org.junit.Test
public void applicationShowsErrorWhenInvalidInputsAreProvided() {
    ChromeDriver driver = createInvisibleChromeDriver();

    driver.get(netflixURL);

    driver.findElement(By.xpath(idInputXPath)).sendKeys(invalidPhoneNumber);
    driver.findElement(By.xpath(passwordInputXPath)).sendKeys(invalidPassword);

    driver.findElement(By.xpath(loginButtonXPath)).click();

    assertThat(driver.findElement(By.xpath(errorMessageXPath)).isDisplayed()).isTrue();
    assertThat(driver.findElement(By.xpath(errorMessageXPath)).getText()).isEqualTo(errorMessageContent);
}
```

4.3. Test Case 3: Successful Login

Unlike test case 2, this test case provides valid information and then tries to log in. ın this way, successful behavior of the application is tested.

```
@org.junit.Test
public void applicationSuccessfullyLogsIn() {
    ChromeDriver driver = createInvisibleChromeDriver();

    driver.get(netflixURL);

    driver.findElement(By.xpath(idInputXPath)).sendKeys(validPhoneNumber);
    driver.findElement(By.xpath(passwordInputXPath)).sendKeys(validPassword);

    driver.findElement(By.xpath(loginButtonXPath)).click();

    assertThat(driver.findElement(By.xpath(successMessageXPath)).isDisplayed()).isTrue();
    assertThat(driver.findElement(By.xpath(successMessageXPath)).getText()).isEqualTo("Successful Login");
}
```

4.4. Test Case 4: Empty Inputs

The application behaves differently when inputs are empty than when inputs are invalid. Therefore, this test case tests whether the application warns the user in a different manner when inputs are empty.

```
@org.junit.Test
public void applicationGivesRelevantFeedbackWhenInputsAreEmpty() {
    ChromeDriver driver = createInvisibleChromeDriver();
    driver.get(netflixURL);

    driver.findElement(By.xpath(loginButtonXPath)).click();

    WebElement errorMessage1 = driver.findElement(By.xpath(errorMessage1XPath));
    WebElement errorMessage2 = driver.findElement(By.xpath(errorMessage2XPath));

    assertThat(errorMessage1.isDisplayed()).isTrue();
    assertThat(errorMessage1.getText()).isEqualTo("Please enter a valid email or phone number.");

    assertThat(errorMessage2.isDisplayed()).isTrue();
    assertThat(errorMessage2.getText()).isEqualTo("Your password must contain between 4 and 60 characters.");
}
```

.

4.5. Test Case 5: Facebook Login

This is the test for facebook login functionality. After clicking the facebook button, it will go to facebook api and connect automatically.

```
@org.junit.Test
public void facebookSuccessfulLogin() throws InterruptedException {
    ChromeDriver driver =  createInvisibleChromeDriver();
    driver.get(facebookURL);
    Thread.sleep( millis: 500);

    driver.findElement(By.xpath(facebookButton)).click();
}
```

5. Automation Experience

Using an automation tool elevates the testing experience to another level in my opinion. It is more professional and efficient compared to manual testing. Although automated testing needs some effort and time; once the tests are written, it takes nothing but just clicking run to test which is drastically beneficial especially when developing enterprise systems.

Moreover, once the tests are written, it is much faster than manual testing, since it is not human to click buttons or enter inputs.

The most interesting experience in this project was establishing 30 connections simultaneously. This is impossible in manual testing. Thus, we have learned that we can test more than we can manually test.

6. Test Automation Benefits in terms of Velocity and Quality

Test automation allows the program to be scrutinized in a fast fashion. Test automation does not require many engineers, rather one engineer creates the scripts/test cases to base the test automation on; this makes the test automation process efficient. Test automation allows the location of flaws which can be eradicated at the initial phase of the project. If such flaws are detected at later stages, quality of the software can be greatly compromised along with the development flow. Pointing out flaws in the program at the right time allows the development process to continue with a flow. Not only this, it is also costly and timely to rectify such flaws in the later stages of development. Nonetheless, test automation can be performed in parallel with many scripts being tested for several projects, and can also be executed multiple times. In essence, test automation is crucial in terms of velocity and quality for the software.

7. References

[1]. https://github.com/maryamShahid/cs458-project-1