Bilkent University

CS 411 - Software Architecture Design

Section - 2

Project - 1

Team Members

Hande Sena Yılmaz 21703465

Ege Kaan Gürkan 21803726

Oğulcan Çetinkaya 21803679

Contents

1. Application and Implementation Details

1.1. Screenshots of the Application

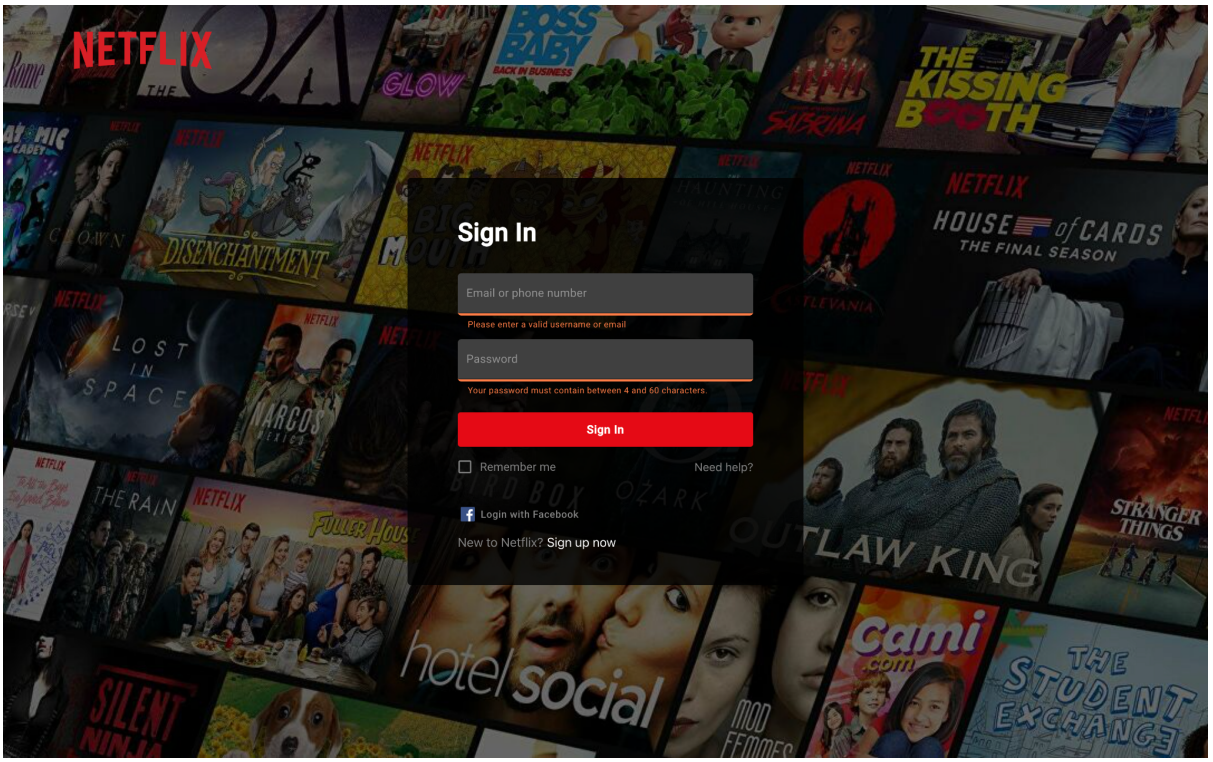


Figure #1

Initial load of the page without any prior "remember me" selection

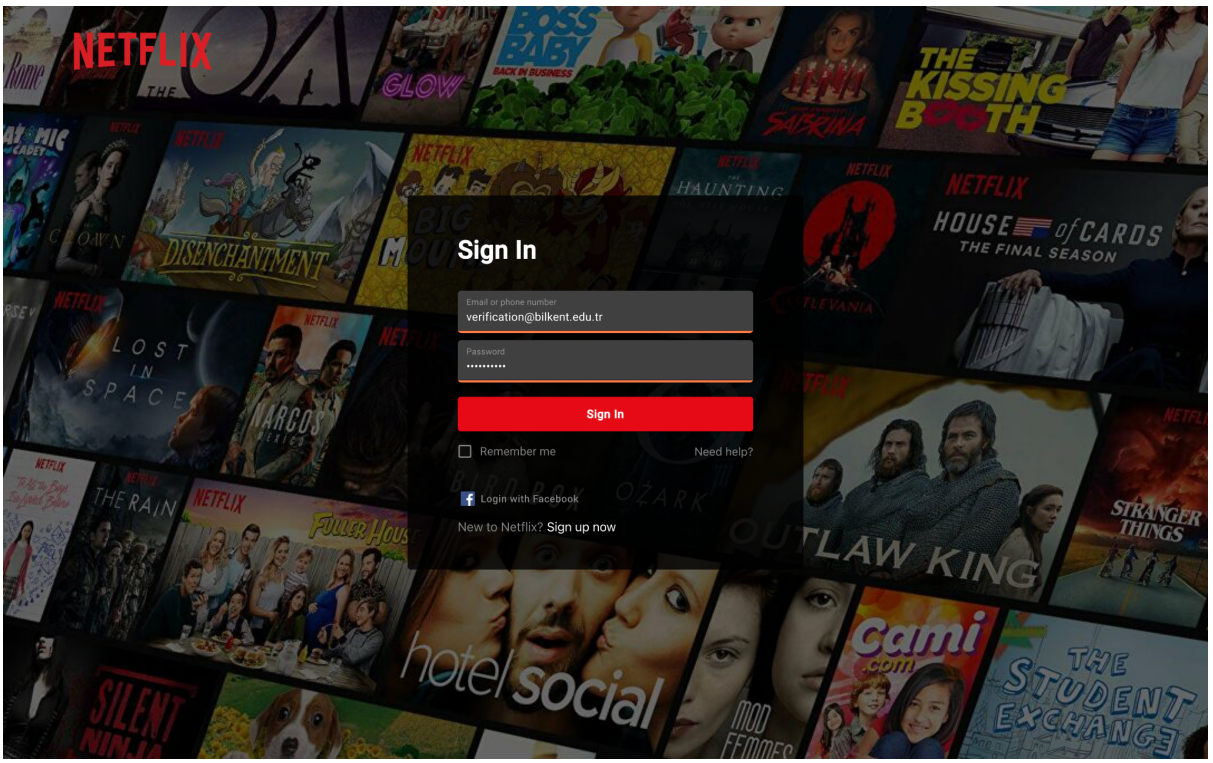


Figure #2

Username and password values entered (warning texts are gone.)

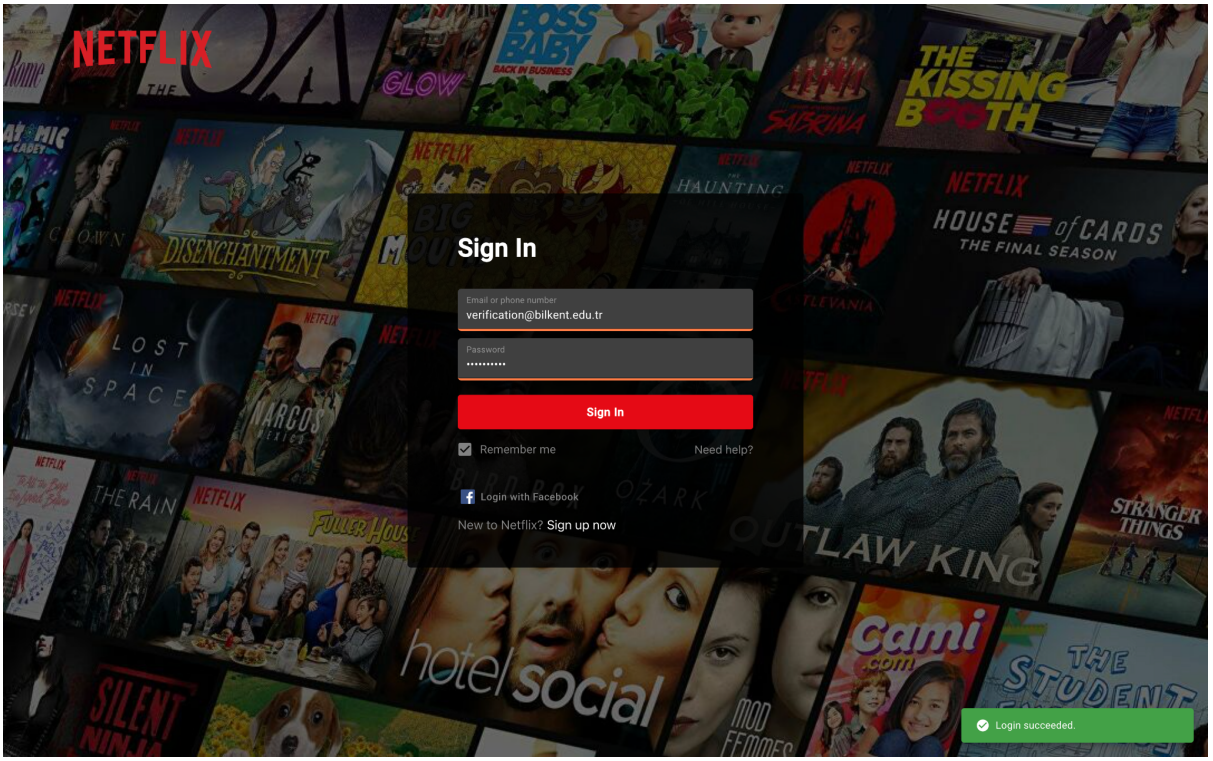


Figure #3

Correct login via the backend API.

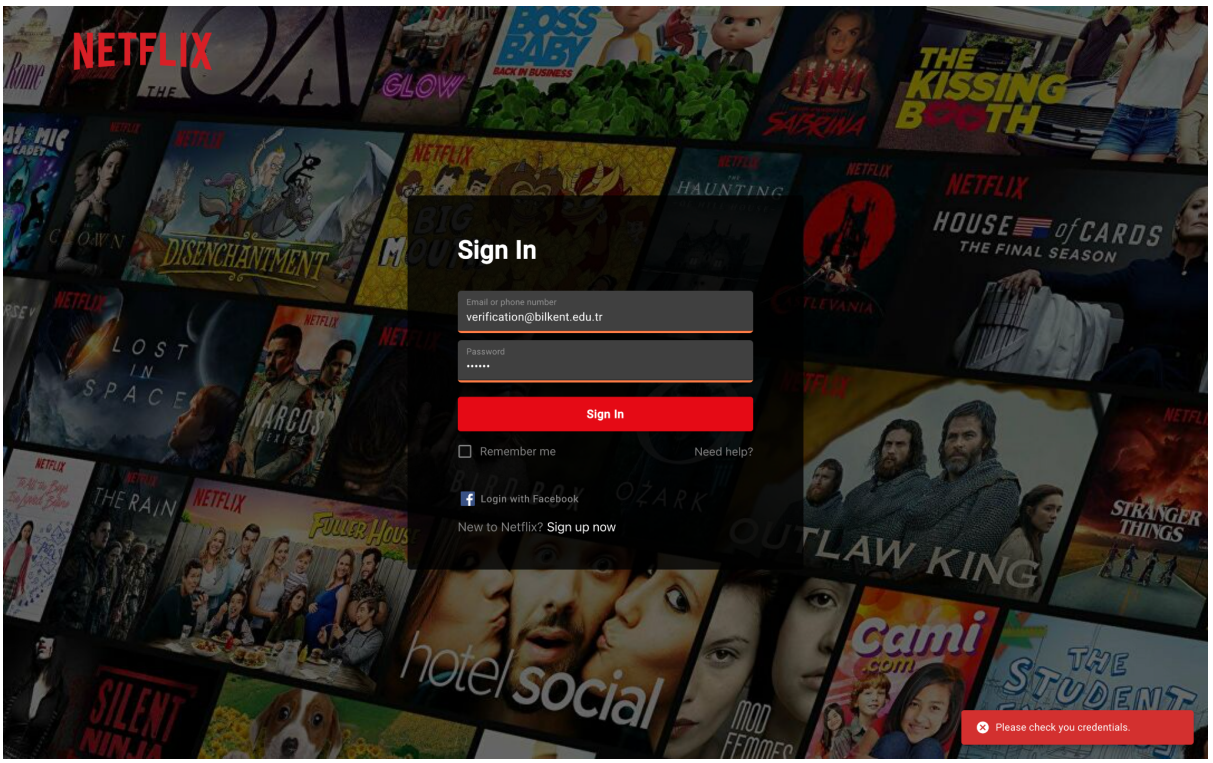


Figure #4

Incorrect login via the backend API.

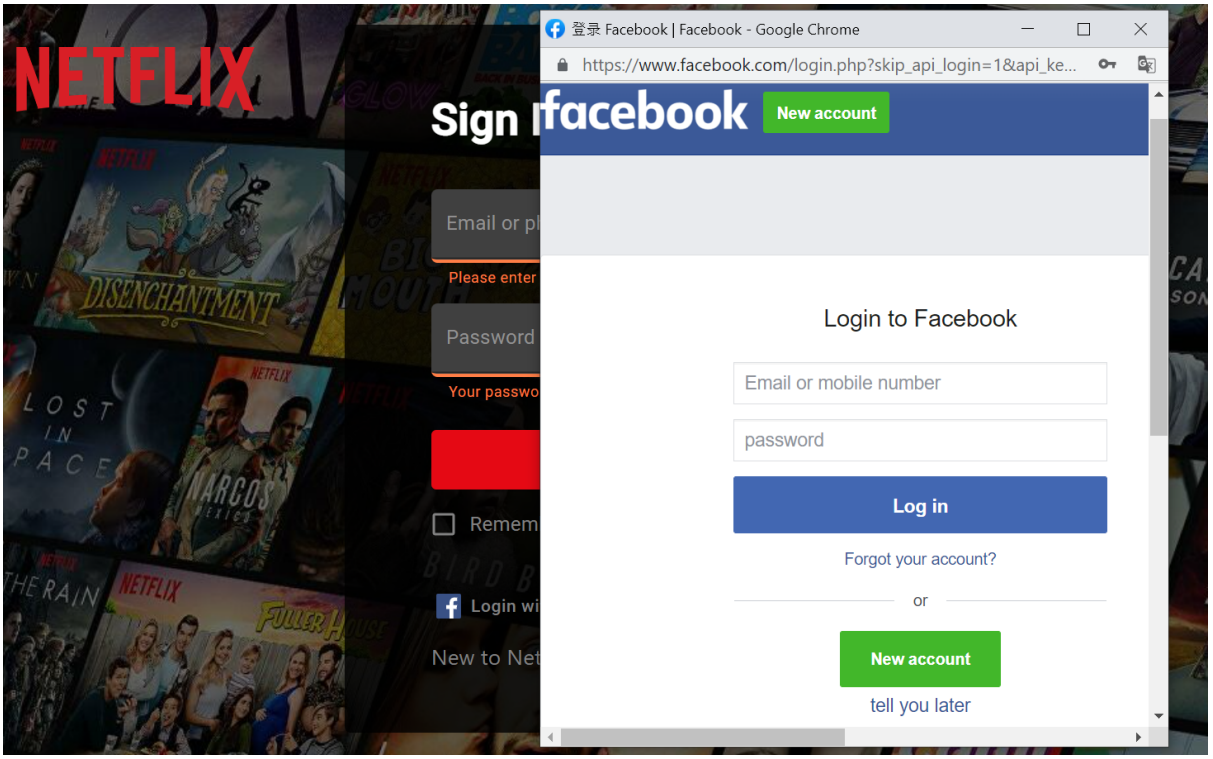


Figure #5

Popup when the "Login with Facebook" button is clicked

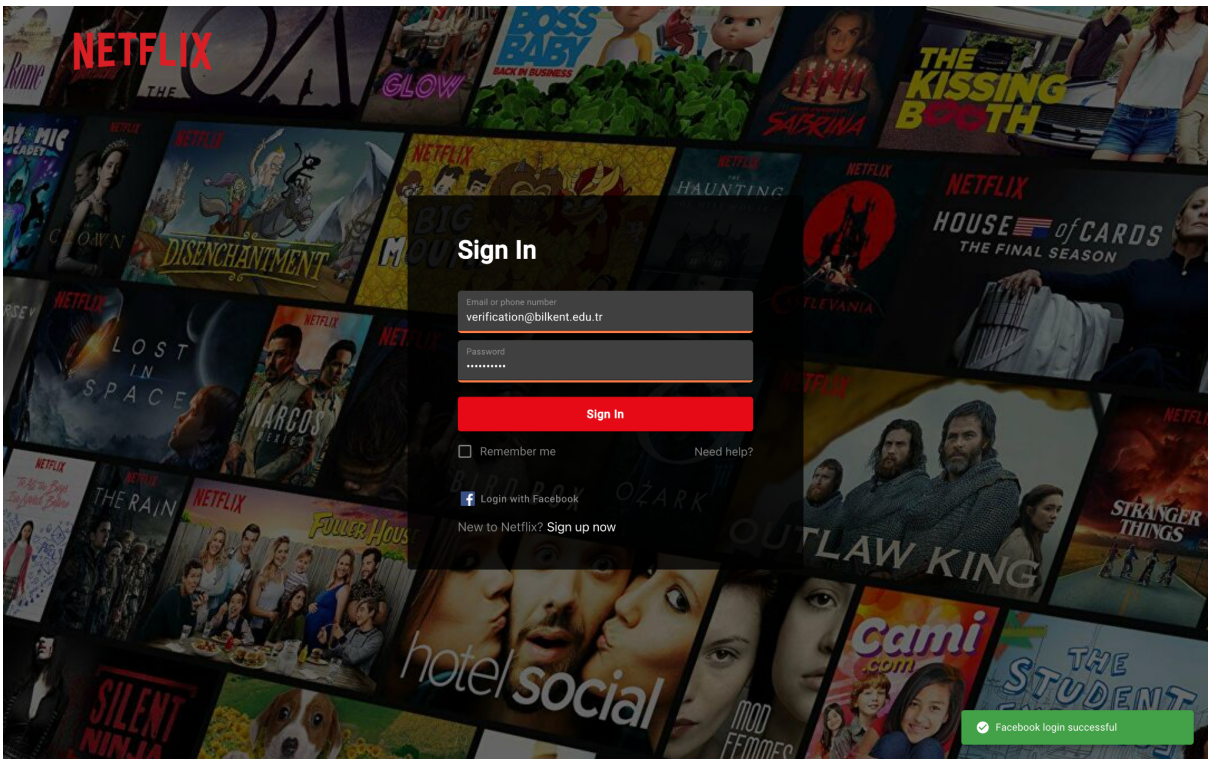


Figure #6

Correct login via the Facebook API.

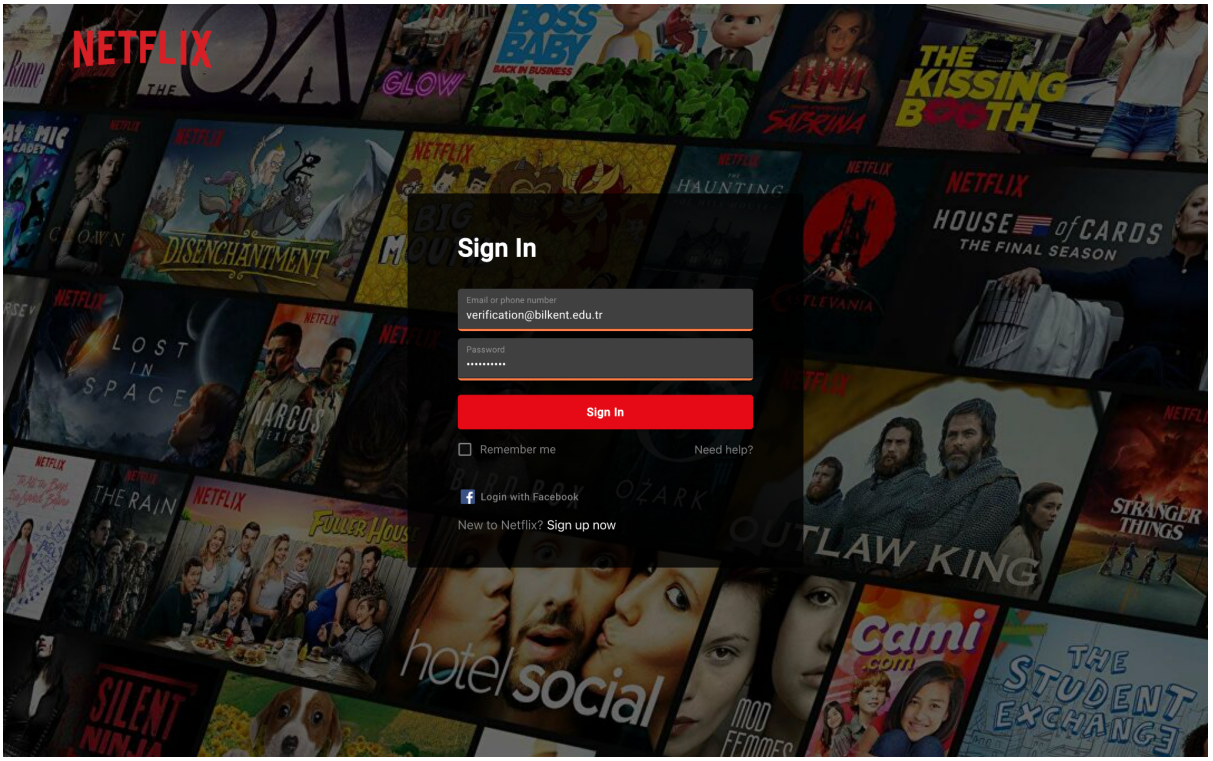


Figure #7

Autofill of the Email and Password fields if there was a successful login with the 'Remember me' checkbox checked.

1.2. Implementation Details

```
// Backend login logic

router.post("/login", (req, res, next) => {

  let sentEmail = req.body.email
  let sentPassword = req.body.password

  let db = JSON.parse(fs.readFileSync(path.resolve(__dirname, "./db.json")))

  if (sentEmail === undefined || sentPassword === undefined || sentEmail === "" || sentPassword === "" )
{   res.status(400).json({
      status: "Error",
      message: "The email or password field was not filled."
    })
  }

  let found = false;

  db.forEach(({email, password}) => {
    if (sentEmail === email && sentPassword === password) {
        found = true;
    }
  })

  if (found) {
    return res.status(200).json({
      status: "OK",
      message: "Login succeeded."
    })
  } else {
    return res.status(401).json({
      status: "Error",
      message: "Please check you credentials."
    })
  }

})}
```

Figure #8, *auth.js*
The logic in the API that controls a successful login.

```
// Frontend submit form logic

const handleSubmit = (event) => {
        event.preventDefault();

        if (displayUsernameHelperText.display || displayPasswordHelperText.display) {
            // Display a Snackbar (warning on the bottom right) component
            // if one of the helper texts are being shown
            enqueueSnackbar("Please fill the username and password fields obeying the rules written beneath the text fields.",
                            {variant: "warning"})
            return
        }

        axios.post(API_URL + "/auth/login", {
            email: formValues.email_or_phone,
            password: formValues.password
        }).then((response) => {

            // If the "remember me" checkbox is selected, save credentials to local storage.
            if (rememberMe) {
                localStorage.setItem("user", JSON.stringify({
                    email: formValues.email_or_phone,
                    password: formValues.password
                }))
            }
            enqueueSnackbar(response.data.message, {variant: "success"})
        }).catch((error) => {
            enqueueSnackbar(error.response.data.message, {variant: "error"})
        })
    }
```

Figure #9, *Login.js*

The logic in the frontend that sends a login request to the API.

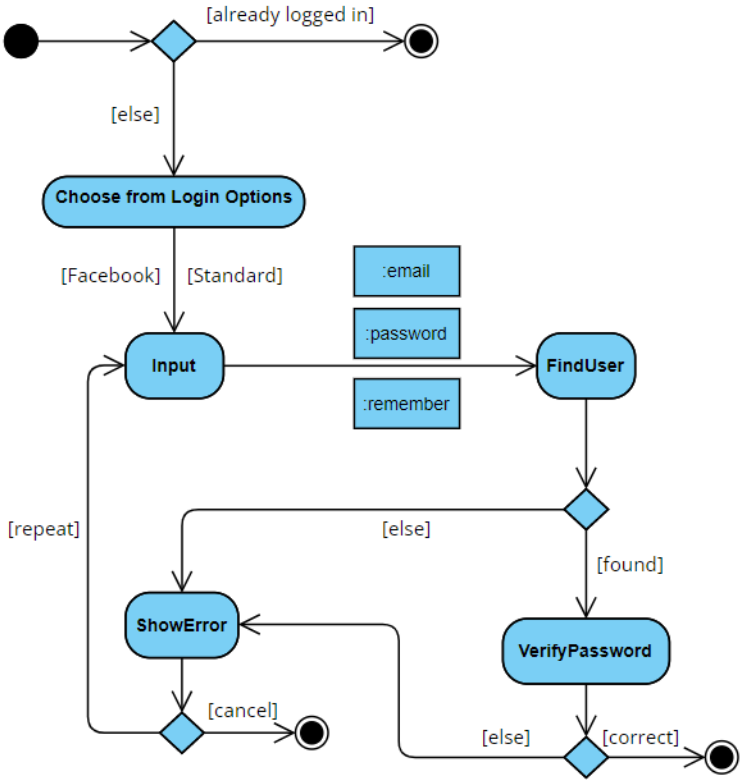
```
function App() {
    return (
        <SnackbarProvider anchorOrigin={{
            vertical: 'bottom',
            horizontal: 'right',
        }}>
            <div className="App">
                <BrowserRouter>
                    <Routes>
                        <Route path={"/"} element={<Navigate to={"/login"}/>}/>
                        <Route path={"/login"} element={<Login/>}/>
                    </Routes>
                </BrowserRouter>
            </div>
        </SnackbarProvider>
    );
}
```

Figure #10, *App.js*

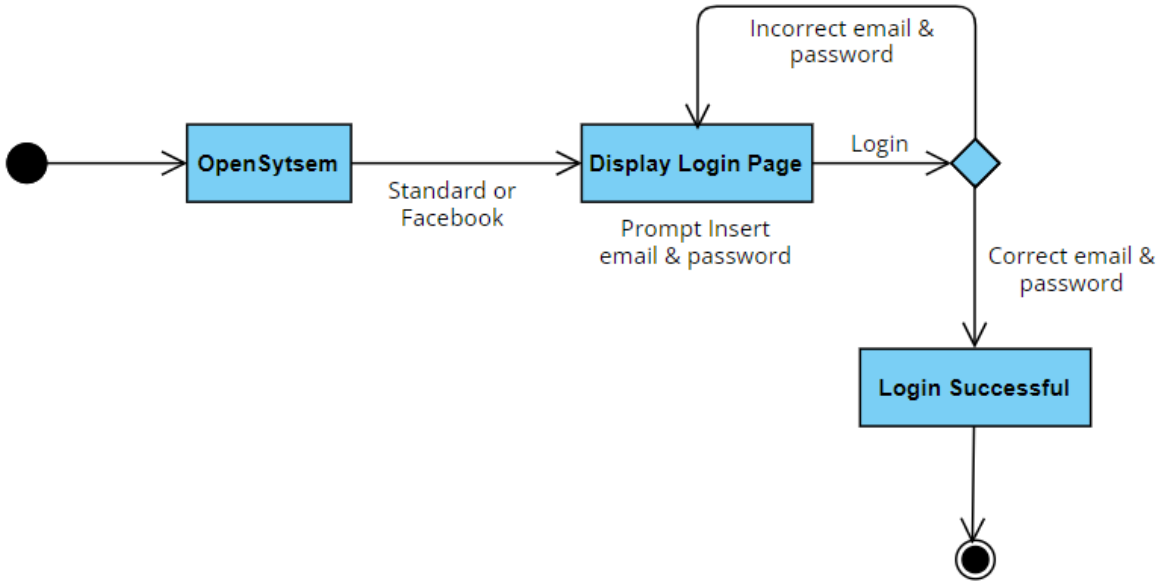
The main router logic of the frontend React App.

2. UML Diagrams

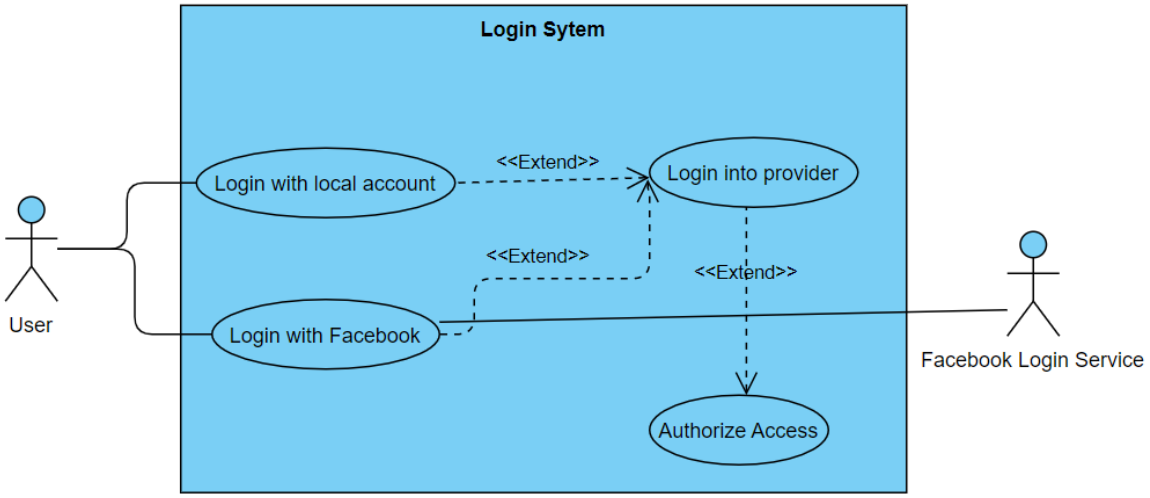
2.1. Activity Diagram



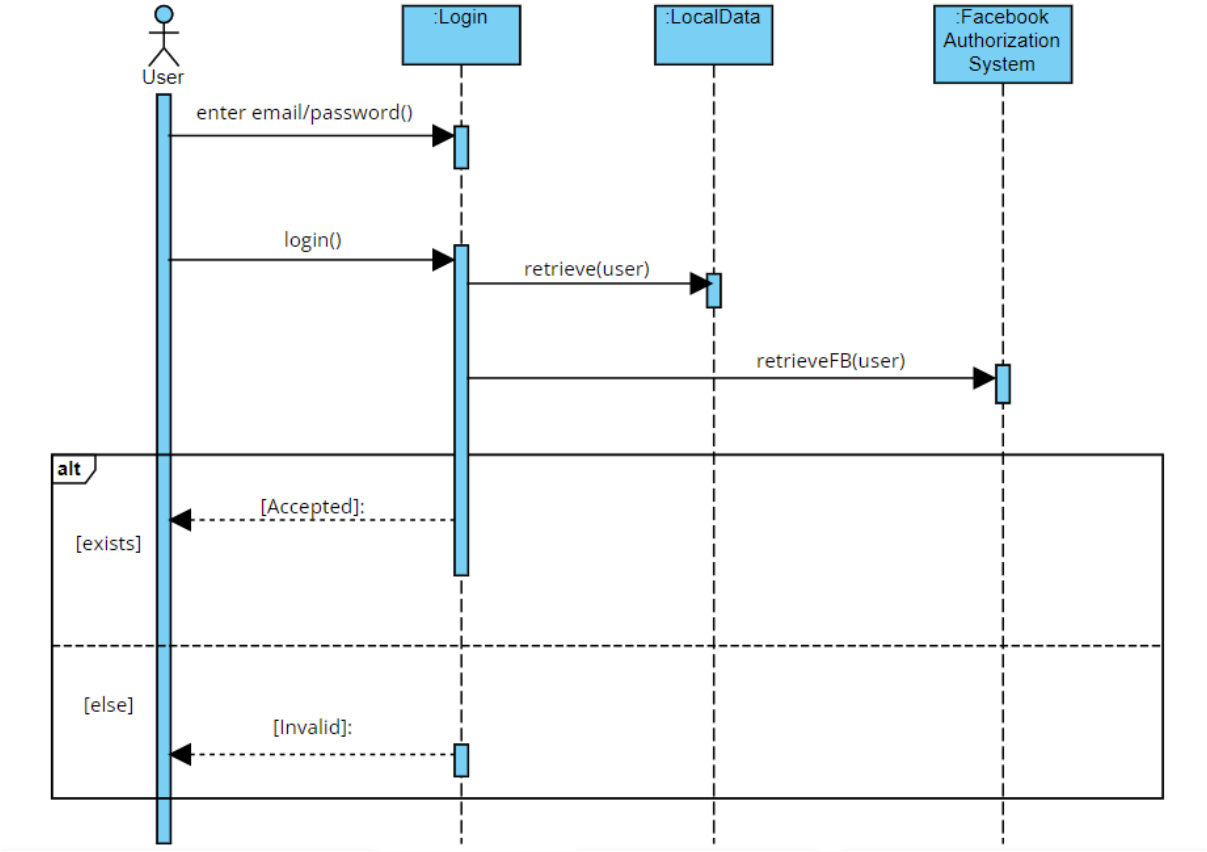
2.2. State Diagram



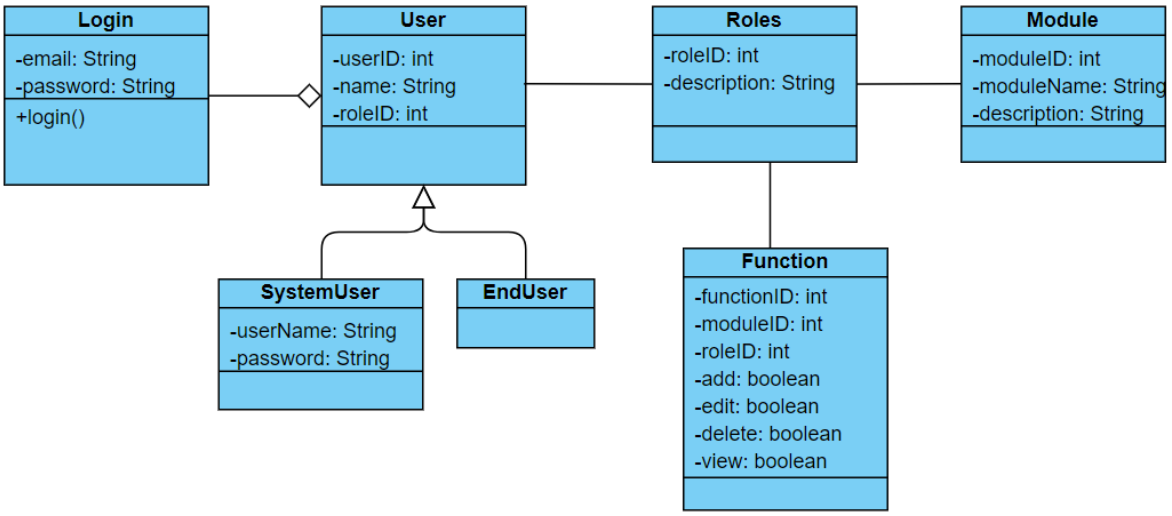
2.3. Use-Case Diagram



2.4. Sequence Diagram



2.5. Class Diagram



3. About Selenium

Selenium is an open-source umbrella project for a range of tools and libraries aimed at supporting web browser automation[1]. Selenium provides a playback tool for authoring functional tests without the need to learn a test scripting language. It also provides a test domain-specific language to write tests in a number of popular programming languages, including JavaScript (Node.js), C#, Java, PHP, Python, Ruby, etc [2]. The tests can then run against most modern web browsers. Selenium runs on Windows, Linux, and macOS. 10 main capabilities of Selenium are the followings [3]:

1. Using “DesiredCapabilities” in WebDriver, it can be set and send the common standard WebDriver capabilities. It is also possible to use the different options classes (“ChromeOptions”, “FirefoxOptions”, “EdgeOptions”, “SafariOptions”, “InternetExplorerOptions”) which have convenient methods for setting the browser-specific capabilities.
2. Automated cross-browser testing can be achieved by passing the “browserName” and “browserVersion” capabilities.
3. It allows creating a proxy object, setting the HTTP or SSL proxies, and sending it with the “proxy” capability. You can also turn off proxy settings and pass it with the capability.
4. It has a headless mode. If we take away the GUI part of the browser and launch it in a non-graphical mode, then the application runs in “*headless browser*”. The advantage of using this to test web applications is that they can do all the things

that normal browsers do, but with improved speed and performance. This is especially helpful for parallel test execution.

5. Testers may need to load an extension (packed or unpacked) or disable an extension while running the automated tests and Selenium allows loading extensions and disabling extensions.
6. It supports the incognito mode in the browsers which erases temporary data collected by the device testing on. It blocks targeted advertisements from showing up when testing the applications. It can also prevent the applications from showing users preferences/suggestions which is required if they want to test the application with its default settings.
7. Sometimes, it may be needed to run the automated tests in various window screen sizes, and Selenium provides setting windows size.
8. It allows setting the default file download location.
9. To handle SSL (Secure Socket Layer) certificate errors and exceptions, Selenium allows testers to use pass “–ignore-certificate-errors” as an argument to the ChromeOptions object.
10. When a new Chrome session gets initiated by the WebDriver, ChromeDriver creates a temporary user profile for that session. During the automated test run, instead of a new user profile, testers might want to use a custom user profile, predefined with special settings like installed extensions, locale, language, etc. They can pass the “user-data-dir” as key and the custom profile path as its value to the capabilities to tell Chrome to use that custom profile for the tests.

4. Test

4.1. Test Cases

To write comprehensive test cases which tests both the functionality and the components of the netflix login page, we decided to form our tests as combinations of UI, functional, non-functional, positive and negative tests.

Test Case 1
The aim of this test case is to check if our website has the required UI components.

1. Verify that the login screen contains elements such as username field, password field, sign in button, remember me checkbox and “Need Help?” link.
2. Verify that all the fields such as username and password has a valid placeholder

Test Case 2

The aim of this test case is to check the functionality of the login. This test case checks whether a user can or cannot login with combinations of valid and invalid usernames and passwords. We check this by writing positive and negative functional tests.

1. Verify that the user is able to login with valid credentials
2. Verify that the use is not able to login with an invalid username and valid password
3. Verify that the user is not able to login with a valid username and invalid password
4. Verify that the user is not able to login with an invalid username and invalid password

Test Case 3
The aim of this test case is to check if the login page is secure enough to not let anyone copy the masked password and see what the real password is.

1. Users should not be allowed to copy and paste passwords from the password field.

Test Case 4
The aim of this test case is to check the login with facebook functionality of the website.

1. Verify that users can login by using the “Login with Facebook” button.

Test Case 5
The aim of this test case is to check if the “Remember-me” button works as intended

1. Verify that the “Remember-me” button works.

To start implementing our test cases, we needed to turn them into test scenarios. The difference between a test case and test scenario is that test scenarios are more detailed and designed in a step-by-step fashion whereas a test case is more general.

Test Scenario 1

1. Go to the URL (http://localhost:3000/login)
2. Check if the username field exists
3. Check if the username field has the correct placeholder
4. Check if the password field exists
5. Check is the password field has the correct placeholder
6. Check if sign in button exists
7. Check if “Remember-me” checkbox exists
8. Check if “Need help?” link exists

Test Scenario 2
To follow the unit testing conventions, the second test case needed to be divided into multiple test scenarios so that no test scenario is too long and each of them tests only specific and one condition.

Test Scenario 2.1
1. Go to the URL (http://localhost:3000/login)

2. Enter a valid username
3. Enter a valid password
4. Click sign in button
5. Check if there is “Login succeeded.” message

Test Scenario 2.2

1. Go to the URL (http://localhost:3000/login)
2. Enter an invalid username
3. Enter a valid password
4. Click sign in button
5. Check if there is “Please check you credentials.” message

Test Scenario 2.3

1. Go to the URL (http://localhost:3000/login)
2. Enter a valid username
3. Enter an invalid password
4. Click sign in button
5. Check if there is “Please check you credentials.” message

Test Scenario 2.4

1. Go to the URL (http://localhost:3000/login)
2. Enter an invalid username
3. Enter an invalid password
4. Click sign in button
5. Check if there is “Please check you credentials.” message

Test Scenario 3

1. Go to the URL (http://localhost:3000/login)
2. Enter a valid password
3. Copy the password in the password field
4. Paste the password into the username field
5. Check if username field contains the valid password

Test Scenario 4

1. Go to the URL (http://localhost:3000/login)
2. Click “Login with Facebook” button
3. Click to the pop-up facebook login page
4. Enter facebook email to e-mail field
5. Enter password to facebook password field
6. Click sign in button
7. Move to the netflix tab
8. Check if the website displays “Facebook login successful” message

Test Scenario 5

1. Go to the URL (http://localhost:3000/login)
2. Enter a valid username to username field
3. Enter a valid password to the password field

4. Click to “Remember-me” checkbox
5. Click sign in button
6. Refresh the browser
7. Check if username field has the valid username
8. Check if password field has the valid password

4.2. Test Implementation Details

We implemented the test cases in Python. We used the built-in unittest framework of Python. Unittest framework in Python uses 2 methods called setup and teardown for each written test case. Before starting each case, setup method is called and initialises the driver. After completing each test case, teardown method is called which closes the chrome driver. The implementations of each test case can be seen below along with setup and teardown methods.

Test Case 1

```
def test_case_1(self):
    #1.2 , 1.3, 1.5, 1.6, 1.8, 1.9, 1.10
    try:
        uname_field = self.driver.find_element(By.ID, "email-or-phone-number")
        self.driver.find_element(By.XPATH, "//label[contains(text(), 'Email or phone number')]")

        pw_field = self.driver.find_element(By.ID, "password")
        self.driver.find_element(By.XPATH, "//label[contains(text(), 'Password')]")

        sign_in_btn = self.driver.find_element(By. XPATH, "//button[contains(text(), 'Sign In')]")

        remember_me = self.driver.find_element(By.NAME, "remember_me")

        need_help = self.driver.find_element(By.XPATH, "//*[contains(text(), 'Need help?')]")

    except NoSuchElementException:
        print("Missing UI component.(Check username/password fields, sign in button ,remember me checkbox or need help link )")
        assert False
```

Test Case 2.1

```
def test_case_2_1(self):
    #2.2
    uname_field = self.driver.find_element(By.ID, "email-or-phone-number")
    #2.3
    uname_field.send_keys("ogulcan@bilkent.edu.tr")
    #2.4
    pw_field = self.driver.find_element(By.ID, "password")
    #2.5
    pw_field.send_keys("secret3")
    #2.6
    sgnin_btn = self.driver.find_element(By. XPATH, "//button[contains(text(), 'Sign In')]")
    sgnin_btn.click()

    #2.7
    try:
        lgn_msg = self.driver.find_element(By.XPATH, "//div[@id='notistack-snackbar' and contains(text(), 'Login succeeded.')]")
        self.assertEqual("Login succeeded.", lgn_msg.text, "Login failed with correct credentials")
    except NoSuchElementException:
        print("Login is not successful")
```

Test Case 2.2

```
def test_case_2_2(self):
    self.driver.find_element(By.ID, "email-or-phone-number").send_keys("invalid@username")

    self.driver.find_element(By.ID, "password").send_keys("verifythis")
    self.driver.find_element(By.XPATH, "//button[contains(text(), 'Sign In')]").click()

    lgn_msg = WebDriverWait(self.driver, 10).until(EC.presence_of_element_located((By.XPATH,
                                                                                   "//div[@id='notistack-snackbar' and contains(text(),
    self.assertEqual("Please check you credentials.", lgn_msg.text, "Login successful with invalid credentials")
```

Test Case 2.3

```
def test_case_2_3(self):
    self.driver.find_element(By.ID, "email-or-phone-number").send_keys("ogulcan@bilkent.edu.tr")

    self.driver.find_element(By.ID, "password").send_keys("invalidpassword")
    self.driver.find_element(By.XPATH, "//button[contains(text(), 'Sign In')]").click()

    lgn_msg = WebDriverWait(self.driver, 10).until(EC.presence_of_element_located((By.XPATH,
                                                                                   "//div[@id='notistack-snackbar' and contains(text(),
    self.assertEqual("Please check you credentials.",  lgn_msg.text, "Login successful with invalid credentials")
```

Test Case 2.4

```
def test_case_2_4(self):
    self.driver.find_element(By.ID, "email-or-phone-number").send_keys("invalid@username")

    self.driver.find_element(By.ID, "password").send_keys("invalidpassword")
    self.driver.find_element(By.XPATH, "//button[contains(text(), 'Sign In')]").click()

    lgn_msg = WebDriverWait(self.driver, 10).until(EC.presence_of_element_located((By.XPATH,
                                                                                   "//div[@id='notistack-snackbar' and contains(text(),
    self.assertEqual("Please check you credentials.", lgn_msg.text, "Login successful with invalid credentials")
```

Test Case 3

```
def test_case_3(self):
    #3.2 , 3.3
    pw_field = self.driver.find_element(By.ID, "password")
    pw_field.click()

    pw_field.send_keys("verifythis")
    #3.4
    pw_field.send_keys(Keys.CONTROL + "a")
    pw_field.send_keys(Keys.CONTROL + "c")

    uname_field = self.driver.find_element(By.ID, "email-or-phone-number")
    uname_field.click()

    uname_field.send_keys(Keys.CONTROL + "v")
    self.assertNotEqual( uname_field.text, "verifythis", "Password can be copied")
```

Test Case 4

```
def test_case_4(self):
    login_fb = self.driver.find_element(By.XPATH, "//button[contains(text(), 'Login with Facebook')]")
    login_fb.click()
    self.driver.implicitly_wait(5)
    orig_wd = self.driver.current_window_handle

    WebDriverWait(self.driver, 5).until(EC.number_of_windows_to_be(2))

    for wd_handle in self.driver.window_handles:
        if wd_handle != orig_wd:
            self.driver.switch_to.window(wd_handle)
            break

    self.driver.find_element(By.ID, "email").send_keys("cojepóó578@naluzotan.com")
    self.driver.find_element(By.ID, "pass").send_keys("123123*")
    self.driver.find_element(By.ID, "loginbutton").click()
    self.driver.switch_to.window(orig_wd)
    lgn_msg = WebDriverWait(self.driver, 10).until(EC.presence_of_element_located((By.XPATH,
                                                                                   "//div[@id='notistack-snackbar' and contains(text(),
```

Test Case 5

```
def test_case_5(self):
    self.driver.find_element(By.ID, "email-or-phone-number").send_keys("ogulcan@bilkent.edu.tr")

    self.driver.find_element(By.ID, "password").send_keys("secret3")

    self.driver.find_element(By.NAME, "remember_me").click()

    self.driver.find_element(By. XPATH, "//button[contains(text(), 'Sign In')]").click()

    self.driver.refresh()

    uname = self.driver.find_element(By.ID, "email-or-phone-number").get_attribute("value")

    pw = self.driver.find_element(By.ID, "password").get_attribute("value")

    self.assertEqual(uname, "ogulcan@bilkent.edu.tr", "Username saved")
    self.assertEqual(pw, "secret3", "Password saved")
```

Setup

```
@classmethod
def setUp(self):
    s = Service("D:/chromedriver.exe")
    self.driver = webdriver.Chrome(service=s)
    self.driver.get("http://localhost:3000/login")
    self.driver.maximize_window()
```

Tear down

```
@classmethod
def tearDown(self):
    self.driver.close()
```

5. Automation Experience

At first it was hard to understand how unittest framework worked. After learning how the unittest framework worked, we learned how to navigate through web page elements. We learned how to construct locators by inspecting a webpage. Being able to locate the web page elements, we then learned how to use the selenium library. The Selenium library enabled us to interact with the web page by using a browser driver (in our case we used chromedriver). Learning the simple methods of selenium to interact with the browser such as “find_element”, “click”, “send_keys” and learning how to use implicit and explicit waits enabled us to implement our test cases. Without the implicit and explicit wait statements, it was hard to interact with the web page due to how fast chromedriver executed our statements. In some cases, our tests would fail not because our login page didn’t meet the requirements but because chromedriver tried to click on an element which the website hadn't loaded yet. We realised how crucial it was to automate the testing process after we finished all of our test cases. This project made us realise that automating testing is crucial even in small (our login page had 5 interactable components) projects. At the end, it took selenium to execute 9 tests in approximately 40-45 seconds. If we didn’t automate the testing cases and continued developing the website, we would need to try these tests by hand which would definitely take more than 45 seconds. One of the benefits of testing is that the test explorer also shows where the test failed. If we were testing by hand, it would be harder for us to understand where the error was and what needed to be fixed. In the end, we learned that the time it takes to automate and implement the test cases is far less than the combined time of the developers who try to test a software by hand. Much time can be saved by automating and implementing test cases.

6. Software Development Lifecycle

Throughout the project, we realised that writing tests is crucial in software development. Even in this small project where one might feel pretty confident that they would not have any bugs, the tests we wrote helped us catch a few related to our system. We realised that we had overlooked the failure case of the Facebook login and were only checking if a response was returned or not, and thus, our system would recognize a failed attempt (which returned an error object) as successful and let users through, only because some object was being returned by Facebook. Another thing we realised was that we did not have a limit on how long the email could be, we were only checking for an at (@) symbol, and doing nothing else to validate the emails on the frontend side. These findings for sure helped the quality of the code and we had consensus that more of these comprehensive tests would help the velocity of the code in the long run, benefiting us in work hours to fix a bug before it even goes into testing by humans rather than a customer finding the bug in production code.

7. References

[1] "The Selenium Browser Automation Project". *selenium.dev*. Retrieved February 28, 2022.

[2] "Selenium IDE Is Dead, Long Live Selenium IDE!". *Selenium IDE Official Blog*. August 6, 2018. Retrieved February 28, 2022 – via seleniumhq.wordpress.com.

[3] TestProject. (2020, November 5). *10 selenium custom capabilities simplifying web testing*. TestProject. Retrieved March 1, 2022, from https://blog.testproject.io/2020/11/05/10-selenium-custom-capabilities-simplifying-web-testing/

[4] Visual Paradigm for UML diagrams.