Übung 1

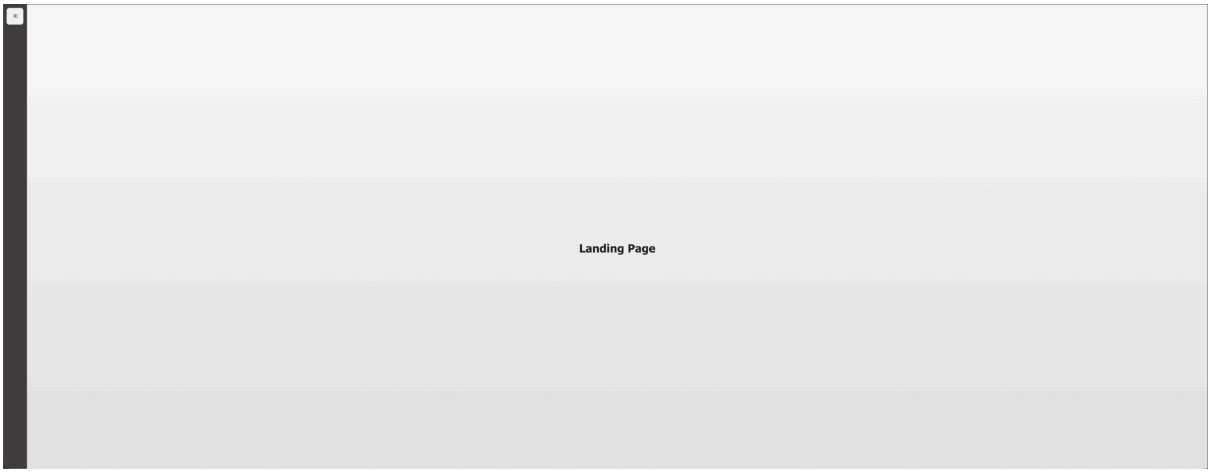
Patrick Kloss,
Matrikelnummer: 107259

Anwendungsbeschreibung mit UML-Diagrammen:

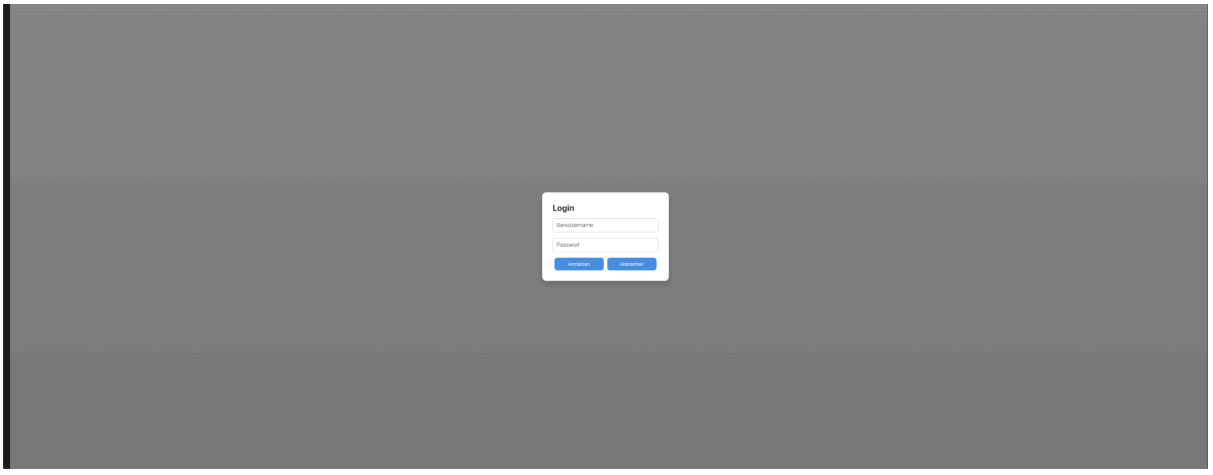
In meiner Anwendung sind die folgenden Tätigkeiten möglich, für die es jeweils auch eine eigene Seite gibt mit variierenden Aktionsmöglichkeiten, je nachdem man Admin oder einfacher User ist.

Die Seiten und Aktionsmöglichkeiten sind wie folgt:

Anmeldeseite (Admin / User):



Hier kann der User sich über den Anmelde-Button (Obere Linke Ecke) anmelden.
Anmelde Button:



Nach dem Anmelden Landet man auf der Starting Page:

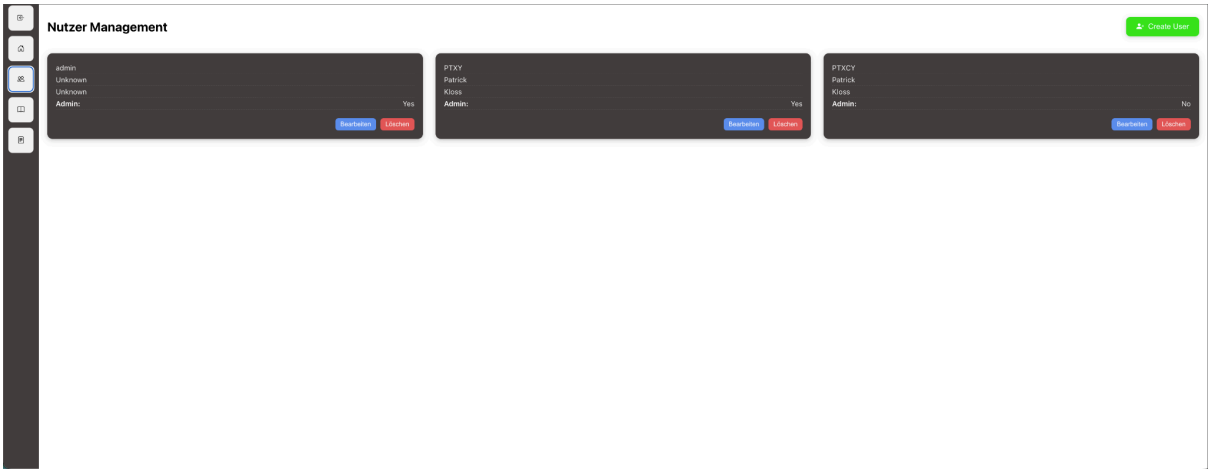
Admin version:

User version:

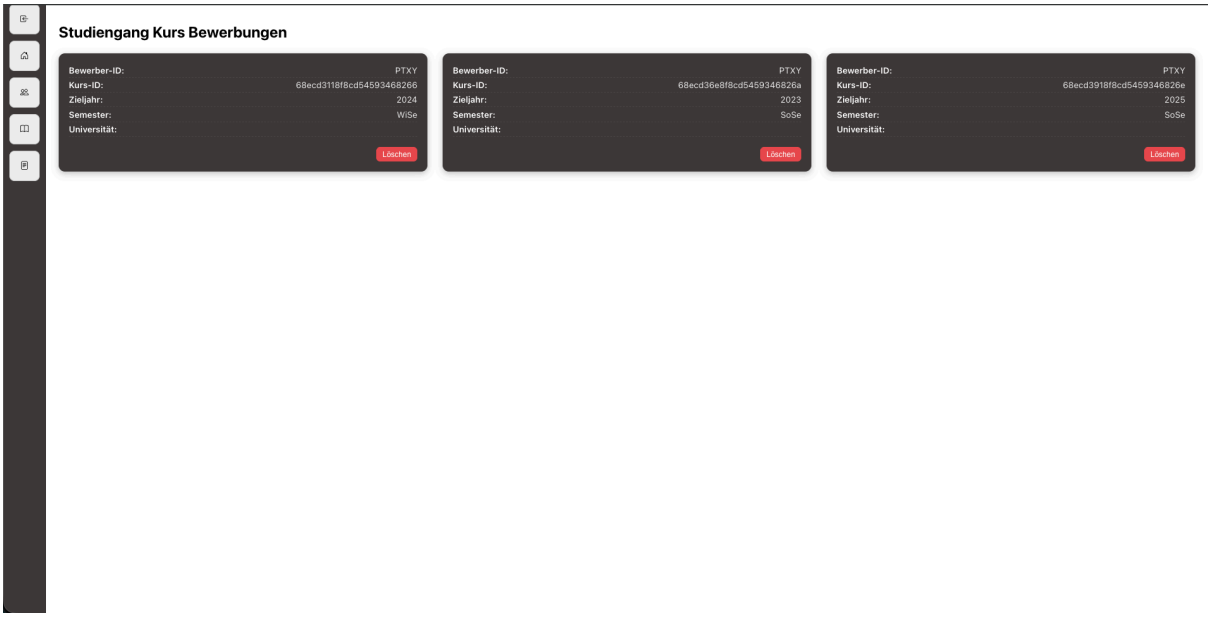
Ab hier ändert sich die Seitennavigation nun für alle Seiten nicht mehr, es ist also das Navigieren zurück zur Startseite und anderen Seiten immer möglich.

Wir können können auf die Folgenden Seiten Navigieren wobei die Nutzerverwaltungs Seite nur für Admins Sichtbar ist:

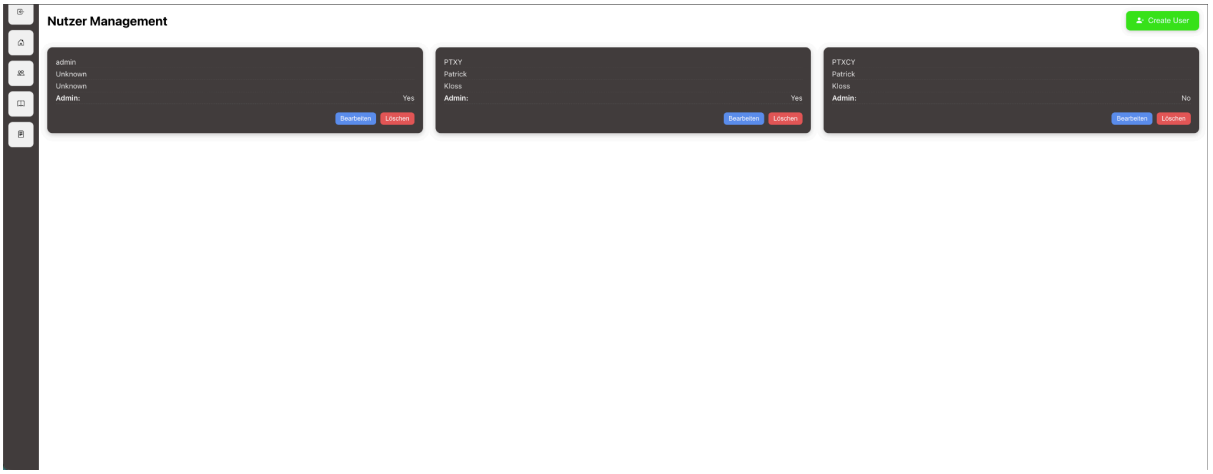
Studienkurse Verwaltungs Seite:



Studienkurs Bewerbungen Verwaltungsseite:



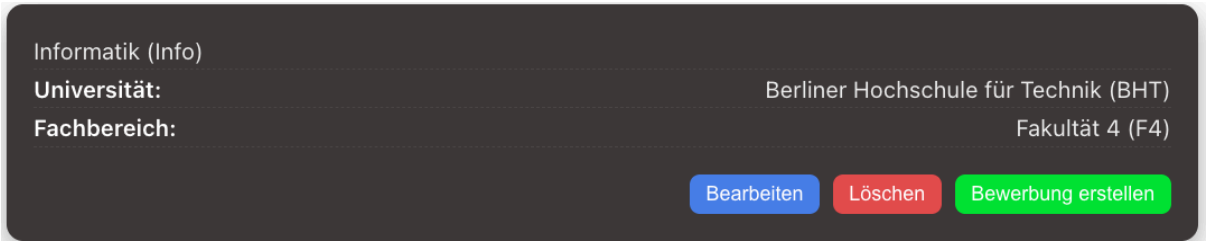
Nutzer Verwaltungsseite:



Hier sind nochmal die wesentlichen Unterschiede zwischen Admin und User Möglichkeiten auf dem Studienkurs und dem Studienkurs Bewerbungen Verwaltungsseiten.

Studienkurse Verwaltungs Seite:

Admin:



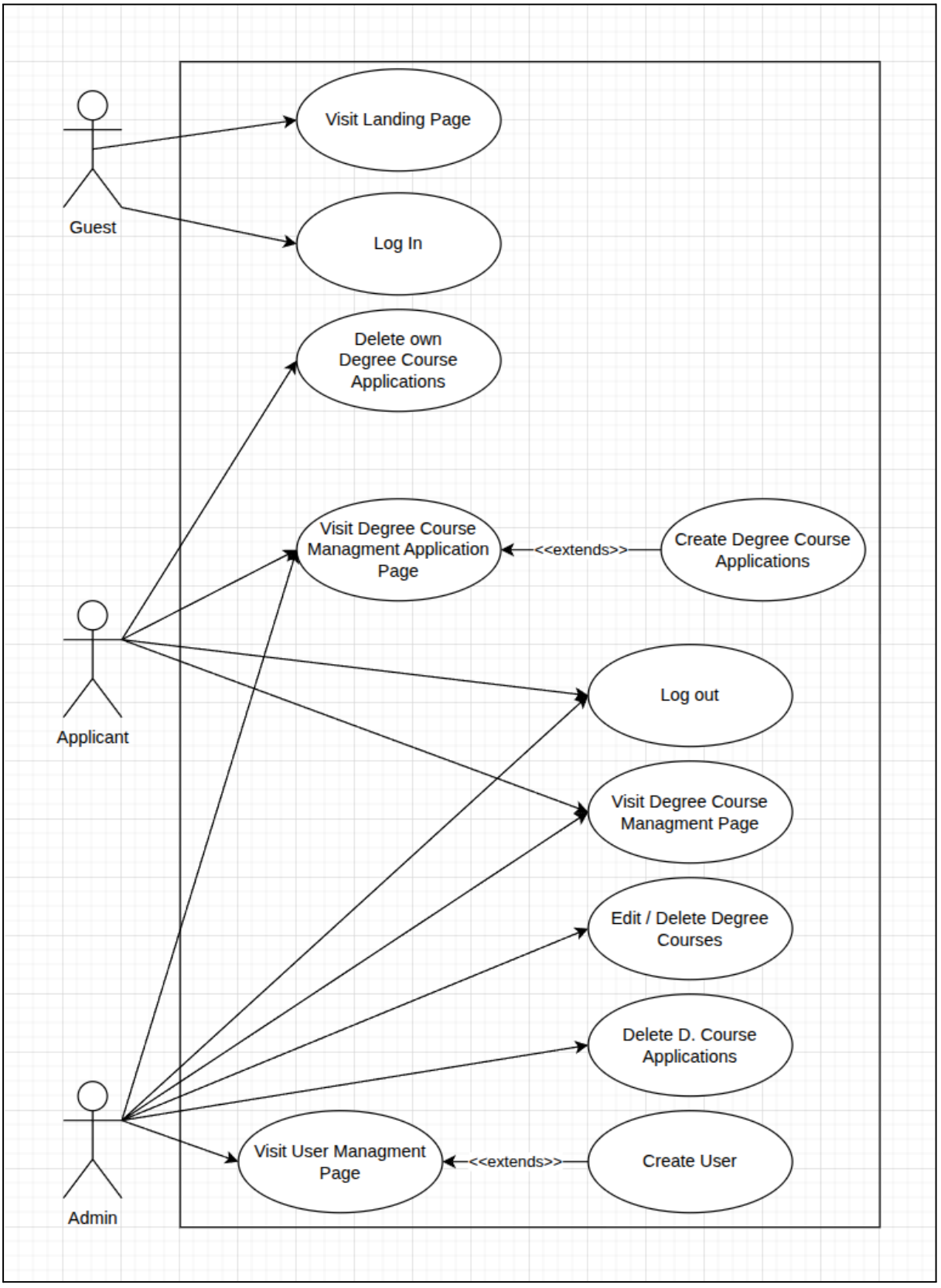
User:

Studienkurs Bewerbungen Verwaltungsseite:

Admin (Sieht Alle Bewerbungen):

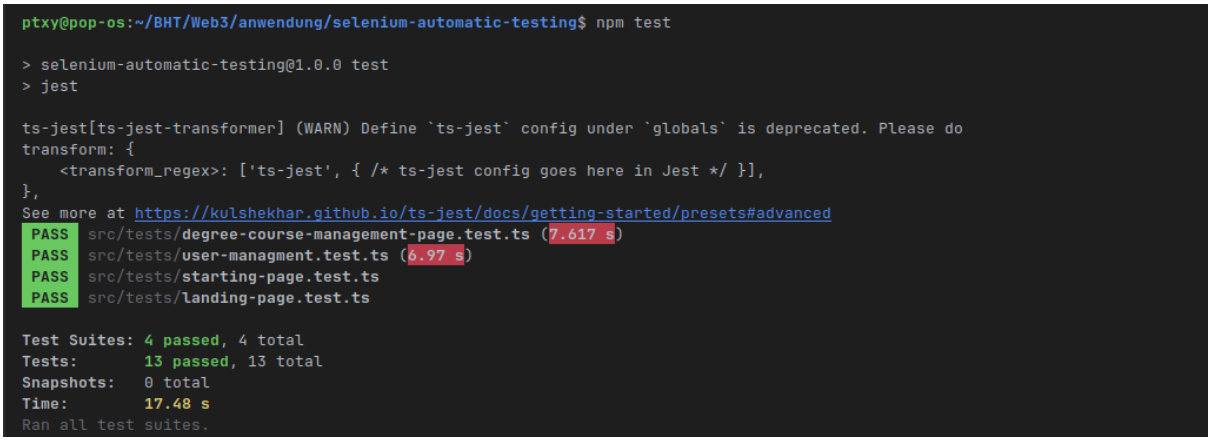
User (Sieht nur die eigenen):

Die vorigen beschreibungen lassen sich dann als folgendes use-case diagramm darstellen:



Automatische Tests:

Die automatisierten Tests wurden mit Jest und Selenium umgesetzt. Im Rahmen der Übung habe ich 4 Jest Test Suits erstellt. Jeweils eine für die Landing und Starting Page, eine für die Anforderungen in der Aufgabe und abschließend eine Zusatzleistung für das Testen der Kurs Management Seite. In der Summe komme ich auf 13 Umfangreiche Tests in den 4 Suits. Die Auswertung sieht etwa wie folgt aus:



Ausgeführt werden diese Test unter dem Subordner “selenium-automatic-testing”. Da ich mein eigenes Backend verwendet habe und daher auch meine eigenen Zertifikate für den Https Prozess. Ich musste auch ein Chrome Profil im Projekt hinterlegen, welches diese Zertifikate installiert hat und kannte, um den Backend-Austausch zu ermöglichen.

Um nochmal auf die Zusatzleistung einzugehen:
Zu finden in „degree-course-management-page.test.ts“, habe ich folgende Tests angelegt:

1. Create / Edit / Delete Course as Admin:
   Simuliert das Anlegen eines neuen Studiengangs, Bearbeiten des Namens und anschließendes Löschen. Alle Eingaben werden über die Benutzeroberfläche durchgeführt und nach jedem Schritt wird überprüft, ob die Änderungen korrekt übernommen bzw. entfernt wurden.

2. Create Non-Admin Account und Prüfung der Berechtigungen:
   Erstellt einen neuen Benutzer und prüft nach dem Login, dass nur die für Nicht-Admins erlaubten Funktionen verfügbar sind. Insbesondere wird sichergestellt, dass der Button zum Erstellen von Studiengängen nicht zugänglich ist.

Bei allen Tests auch außerhalb der Zusatzleistung werden die Browser Logs im Falle eines fehlgeschlagenen Tests ausgegeben.

Hier können sie noch einmal eine hand von kleineren Beispielen sehen wie ich die test etwa umgesetzt habe:

```
test( name: "Check if Admin is Listed in the User Management page", async () :Promise<void> => {
    //TODO MAKE SAVE LATER WHEN PIPELINING IS ADDED
    const username = "admin"
    const password = "123"
    await logIn(driver,username,password);
    try {
        const openUserManagementPageButton :WebElement = await driver.findElement(By.css( selector: "#OpenUserManagementPageButton"));
        await openUserManagementPageButton.click()
        await driver.findElement(By.css( selector: "#UserItemadmin"));
        // no error means elements exist
    } catch (err) {
        const logs :Entry[] = await driver.manage().logs().get(logging.Type.BROWSER);
        console.error( message: "Browser logs during failed test:");
        logs.forEach(entry :Entry => console.error( message: `[${entry.level.name}] ${entry.message}`));
        throw err;
    }
})
```

```
test( name: "Check if Admin can create a user and if new user is listed directly", async () :Promise<void> => {
    const username = "admin"
    const password = "123"
    await logIn(driver,username,password);
    try {
        const openUserManagementPageButton :WebElement = await driver.findElement(By.css( selector: "#OpenUserManagementPageButton"));
        await openUserManagementPageButton.click()

        const createUserButton :WebElement = await driver.findElement(By.css( selector: "#UserManagementPageCreateUserButton"));
        await createUserButton.click();

        const usernameInput :WebElement = await driver.findElement(By.css( selector: "#CreateUserComponentEditUserID"));
        const firstNameInput :WebElement = await driver.findElement(By.css( selector: "#CreateUserComponentEditFirstName"));
        const lastNameInput :WebElement = await driver.findElement(By.css( selector: "#CreateUserComponentEditLastName"));
        const passwordInput :WebElement = await driver.findElement(By.css( selector: "#CreateUserComponentEditPassword"));
        const createButton :WebElement = await driver.findElement(By.css( selector: "#CreateUserComponentCreateUserButton"));

        await usernameInput.sendKeys( ...var_args: "newUser");
        await firstNameInput.sendKeys( ...var_args: "New");
        await lastNameInput.sendKeys( ...var_args: "User");
        await passwordInput.sendKeys( ...var_args: "Xc7@bY5!nR2#tK9q");
        await createButton.click();

        await driver.sleep( ms: 300);
        await driver.findElement(By.css( selector: "#UserItemnewUser"));
    } catch (err) {
        const logs :Entry[] = await driver.manage().logs().get(logging.Type.BROWSER);
        console.error( message: "Browser logs during failed test:");
        logs.forEach(entry :Entry => console.error( message: `[${entry.level.name}] ${entry.message}`));
        throw err;
    }
})

test( name: "Check if created user can be used to logIn and not see the user management page", async () :Promise<void> => {
    const username = "newUser"
    const password = "Xc7@bY5!nR2#tK9q"
    await logIn(driver,username,password);
    try {
        const heading :WebElement = await driver.findElement(By.css( selector: "h1"));
        expect(await heading.getText()).toBe( expected: "Starting Page");
        await expect(async () :Promise<void> => {
            await driver.findElement(By.css( selector: "#OpenUserManagementPageButton"));
        }).rejects.toThrow();

    } catch (err) {
        const logs :Entry[] = await driver.manage().logs().get(logging.Type.BROWSER);
        console.error( message: "Browser logs during failed test:");
        logs.forEach(entry :Entry => console.error( message: `[${entry.level.name}] ${entry.message}`));
        throw err;
    }
})
```

Fehlerbehebung / Code Formatting / Lessons Learned:

Abschließend habe ich noch ein paar Kleinigkeiten in meinem UI verbessert, hauptsächlich Spacing und Padding, die mir vorher nicht aufgefallen waren. Am Backend musste ich nicht mehr arbeiten. Zum Formatieren habe ich Prettier für das neue Selenium-Testing-Projekt installiert. Schwierig war vor allem, die Bugs zu beseitigen, die durch das asynchrone Laden vom Backend entstanden sind. Zum Beispiel: Ich klicke einen Button, und durch “awawaitait” ist man nicht abgesichert, dass die neue View schon gerendert ist. Ansonsten habe ich gelernt, wie man E2E-Tests strukturiert aufbaut, und interessant war für mich besonders das Einbinden der Browser-Logs in den Code.