

# **Test av IT-system**

Individuell uppgift 3: Automatiserade UI-tester

Luleå Tekniska Universitet

I0015N

VT25

Noah Alvandi



# Innehållsförteckning

Innehållsförteckning.....	2
Introduktion.....	3
Testplan.....	4
Resultat.....	7
Reflektion.....	9
Förbättringsförslag.....	9
Identifierade brister.....	10
Slutsats.....	10

# Introduktion

Bothniabladet är en digital bildbyrå som hanterar ett omfattande bildarkiv för interna och externa användare. För att stödja det dagliga arbetet med att hantera bilder, söka efter material och säkerställa rättighetshantering har ett nytt webbaserat system utvecklats. Det är avgörande att detta system fungerar stabilt, är användarvänligt och klarar av olika användarscenarier. Därför har automatiserade UI-tester tagits fram som ett led i kvalitetssäkringen. Denna rapport redovisar design och implementering av sådana tester, där fokus har legat på två centrala funktioner: inloggning och bildsökning. Genom att använda moderna testverktyg som Playwright, Cypress och Selenium IDE har testscenarier skapats som simulerar användarbeteende och verifierar systemets respons vid både typiska och avvikande situationer.

# Testplan

För att testa systemet användes tre olika verktyg: Playwright, Cypress och Selenium. Samtliga verktyg tillämpade i grunden samma testlogik och scenarioflöden men implementerades med olika tekniska ramverk.

Playwright installerades via kommandot:

```
npm init playwright@latest
```

och testfall kördes genom:

```
npx playwright test
```

Cypress användes också som ett kodbaserat projekt där alla tester skrevs i TypeScript. Inga GUI-verktyg eller tillägg användes. Istället exekverades testerna genom

```
npx cypress run
```

eller motsvarande terminalkommandon för att simulera autentiska användarscenarier via kod (Jag använder IntelliJ Idea utvecklingsverktyg så jag skrev inget kommando och jag tryckte bara på spel-knappen för att köra koden).

Selenium användes som ett kodbaserat projekt skrivet i Java. Projektet strukturerades i enlighet med vanliga testautomatiseringsprinciper där testfall implementerades med hjälp av Selenium WebDriver och JUnit. Genom att exekvera dessa Java-baserade tester kunde samma funktionaliteter verifieras med hjälp av kod som anropar webbläsaren direkt.

Nedan presenteras testfallen fördelade över två huvudområden: inloggning och bildsökning. Varje testfall dokumenterar syftet med testet, den testdata som använts, förväntat resultat, faktiskt resultat vid körning samt teststatus (tabeller 1 och 2).

**Tabell 1.** Testfallen till inloggningsfunktionalitet.

Testfall	Syfte	Testdata	Förväntat resultat	Faktiskt resultat	Status
TC01	Korrekt inloggning (Stina)	Användarnamn: stina, Lösenord: fåGelskådning	Redirect till bildsökning sidan	Bildsökning sidan	OK
TC02	Korrekt inloggning (Johan)	Användarnamn: johan, Lösenord: FotoGrafeRing1!	Redirect till bildsökning sidan	Bildsökning sidan	OK
TC03	Fel lösenord	Användarnamn: stina, Lösenord: wrongPassword	Felmeddelande visas	"Invalid username or password" visas	OK
TC04	Okänd användare	Användarnamn: nonexistentuser, Lösenord: somePassword	Felmeddelande visas	"Invalid username or password" visas	OK
TC05	Endast användarnamn	Användarnamn: stina, Lösenord: [tomt]	Fokus förblir i lösenordsfältet	Fokus kvar i lösenordsfält	OK
TC06	Endast lösenord	Användarnamn: [tomt], Lösenord: fåGelskådning	Fokus förblir i användarnamn	Fokus kvar i användarnamn	OK
TC07	Tomma fält	Användarnamn: [tomt], Lösenord: [tomt]	Fokus i användarnamn	Fokus kvar i användarnamn	OK

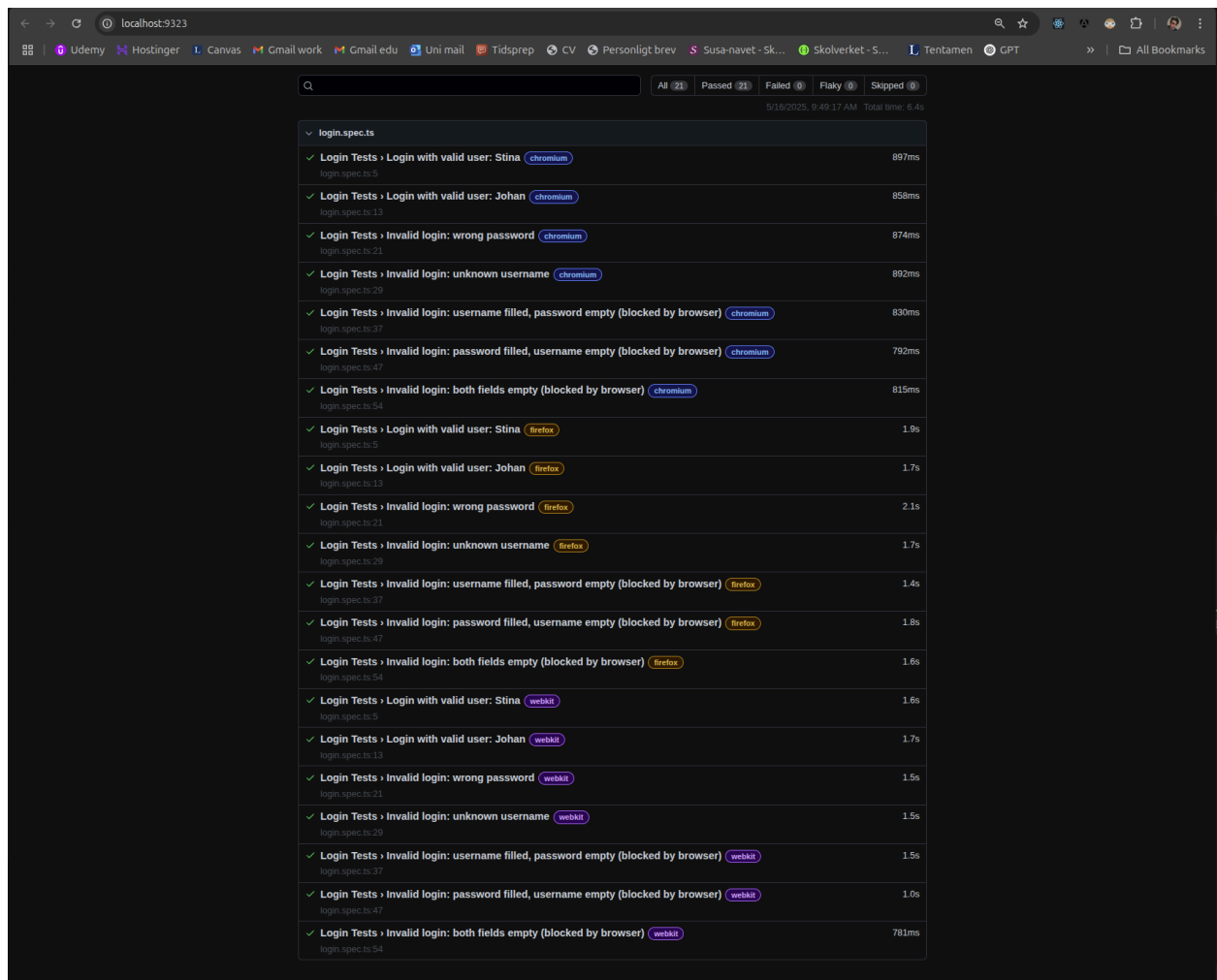
**Tabell 2.** Testfallen till bildsökningssystemet.

Testfall	Syfte	Testdata	Förväntat resultat	Faktiskt resultat	Status
TC08	Sökning med korrekt nyckelord	Sökterm: Empire (eller annan från API)	Bilder returneras	Bilder visas	OK
TC09	Sökning med versaler	Sökterm: EMPIRE	Bilder returneras	Bilder visas inte	FAIL
TC10	Sökning med ogiltigt ord	Sökterm: invalidsearch123456	"Inga träffar"-meddelande visas	Meddelande visas korrekt	OK
TC11	Tom söksträng	Sökterm: [tomt]	Fält kvar synligt, ingen krasch	Ingen krasch sker	OK

# Resultat

För varje test implementerades kod i Playwright, Cypress, och Selenium för att simulera användarinteraktion och bekräfta resultat. Exempel på dessa testfall återges i figurer 1 och 2 samt koden som bifogats zip-filen, och matchar exakt de scenarier som dokumenterats i tabellerna ovan.

Testfallen visar att flertalet funktionaliteter fungerar enligt specifikation. Enda negativa resultatet uppstod vid sökning med versaler, vilket antyder att systemet är skiftlägeskänsligt trots att det inte borde vara det.



The screenshot displays the Playwright test results interface. At the top, a search bar and summary statistics are visible: 'All (23)', 'Passed (23)', 'Failed (0)', 'Flaky (0)', and 'Skipped (0)'. Below this, a table lists individual test cases grouped by browser. Each row shows a green checkmark for a passed test, the test name, the browser used, and the execution time.

Browser	Test Case	Time
Chromium	Login Tests > Login with valid user: Stina	897ms
	Login Tests > Login with valid user: Johan	858ms
	Login Tests > Invalid login: wrong password	874ms
	Login Tests > Invalid login: unknown username	892ms
	Login Tests > Invalid login: username filled, password empty (blocked by browser)	830ms
	Login Tests > Invalid login: password filled, username empty (blocked by browser)	792ms
	Login Tests > Invalid login: both fields empty (blocked by browser)	815ms
Firefox	Login Tests > Login with valid user: Stina	1.9s
	Login Tests > Login with valid user: Johan	1.7s
	Login Tests > Invalid login: wrong password	2.1s
	Login Tests > Invalid login: unknown username	1.7s
	Login Tests > Invalid login: username filled, password empty (blocked by browser)	1.4s
	Login Tests > Invalid login: password filled, username empty (blocked by browser)	1.8s
	Login Tests > Invalid login: both fields empty (blocked by browser)	1.6s
WebKit	Login Tests > Login with valid user: Stina	1.6s
	Login Tests > Login with valid user: Johan	1.7s
	Login Tests > Invalid login: wrong password	1.5s
	Login Tests > Invalid login: unknown username	1.5s
	Login Tests > Invalid login: username filled, password empty (blocked by browser)	1.5s
	Login Tests > Invalid login: password filled, username empty (blocked by browser)	1.0s
	Login Tests > Invalid login: both fields empty (blocked by browser)	781ms

**Figur 1: Testresultat för inloggningsfunktionalitet i Playwright**

Denna bild visar testresultatet för inloggningsscenarier, inklusive både giltiga och ogiltiga inloggningsförsök. Alla testfall har passerat i tre olika webbläsare (Chromium, Firefox och WebKit), vilket tyder på att inloggningslogiken fungerar korrekt oberoende av användarscenario och webbläsartyp.

Test Name	Status	Browser	Time
Image Search Tests › Search is case insensitive	Failed	chromium	5.8s
Image Search Tests › Search is case insensitive	Failed	firefox	6.6s
Image Search Tests › Search is case insensitive	Failed	webkit	6.4s
Image Search Tests › Search for a real keyword from API	Passed	chromium	869ms
Image Search Tests › Invalid keyword returns no results	Passed	chromium	856ms
Image Search Tests › Empty search input does not crash	Passed	chromium	807ms
Image Search Tests › Search for a real keyword from API	Passed	firefox	1.3s
Image Search Tests › Invalid keyword returns no results	Passed	firefox	1.3s
Image Search Tests › Empty search input does not crash	Passed	firefox	1.3s
Image Search Tests › Search for a real keyword from API	Passed	webkit	1.5s
Image Search Tests › Invalid keyword returns no results	Passed	webkit	1.3s
Image Search Tests › Empty search input does not crash	Passed	webkit	1.4s

**Figur 2: Testresultat för bildsökning i Playwright**

Skärmdumpen visar resultatet av automatiserade tester för bildsökning i samtliga webbläsare. De flesta testerna passerar, men testet för versalkänslig sökning misslyckas i samtliga webbläsare, vilket indikerar att funktionen är skiftlägeskänslig – ett oönskat beteende.

För att ytterligare stärka denna rapport har ett videoklipp producerats som visar hur de automatiserade testerna körs. Videon visar exekvering av både inloggnings- och bildsökningstester i verktygen Playwright, Cypress, och Selenium. Genom att visualisera testresultaten i realtid får man en tydlig förståelse för systemets beteende. Länken till videon kommer att bifogas nedan som ett olistat YouTube-klipp som endast är åtkomligt för den som har tillgång till länken.

Länk till videoklipppet: <https://youtu.be/28rITeboxYE>



# Reflektion

Arbetet med automatiserade UI-tester har varit mycket givande och har visat på flera praktiska fördelar. Tester som en gång skrivits kunde köras upprepade gånger utan manuell inblandning, vilket sparade tid och ökade tillförlitligheten i valideringen av funktioner. Den största vinsten låg i möjligheten att snabbt identifiera fel vid kod-förändringar, särskilt vid regressionstestning. Det blev också tydligt hur automatiserade tester kan öka systemets testtäckning, då även mindre vanliga användarscenarier kan testas systematiskt.

Samtidigt medför automatisering vissa utmaningar. UI-ändringar kan snabbt leda till brutna tester om selektorer inte är stabilt definierade. Därför krävs en viss disciplin i både utvecklings- och testdesign för att testerna ska vara långsiktigt hållbara. Det krävdes också viss initial arbetsinsats för att sätta upp och förstå de olika testverktygen, särskilt för Selenium som i detta fall användes som kodbaserat ramverk i Java, vilket fungerar något annorlunda än TypeScript-baserade verktyg som Playwright och Cypress. Sammantaget var erfarenheten mycket lärorik och visade på värdet av testautomatisering i moderna utvecklingsprojekt.

# Förbättringsförslag

Baserat på de utförda testerna och den interaktion som simulerats med användargränssnittet, finns det flera åtgärder som kan vidtas för att göra systemet mer testbart och robust. Det första förbättringsområdet gäller användningen av HTML-elementens attribut. Genom att införa unika ID:n och konsekventa klassnamn för viktiga komponenter blir det enklare att skriva stabila testfall och undvika brutna selektorer vid UI-förändringar.

Vidare skulle gränssnittet kunna göras mer tillgängligt och tydligt genom att förbättra felmeddelanden och valideringslogik. I nuläget ges exempelvis inte alltid tillräcklig återkoppling vid inmatningsfel, vilket försvårar testbarheten och kan skapa förvirring för slutanvändare. Särskilt vid inloggning är det viktigt att fälten reagerar tydligt och logiskt på felaktig eller tom input.

En annan aspekt som identifierats är att sökfunktionen bör göras skiftlägesokänslig. I nuläget påverkas resultaten av huruvida användaren skriver med stora eller små bokstäver, vilket inte är intuitivt. Att åtgärda detta skulle förbättra användarupplevelsen och minska risken för felaktig bedömning av testresultat.

Sammanfattningsvis handlar förbättringsförslagen om att förenkla testskrivning, förbättra användarinteraktionen samt öka systemets robusthet mot varierande inmatning och användarbeteende.

# Identifierade brister

Ett konkret fel som upptäcktes är att sökfunktionen inte är skiftlägesoberoende. Detta bryter mot förväntad funktionalitet då det är rimligt att användare inte ska behöva ta hänsyn till gemener/versaler i fritext sökningar. Ett annat problem gäller kontrastfärg för felmeddelanden, vilket kan strida mot tillgänglighetsriktlinjer. Slutligen noterades viss försening i rendering av felmeddelanden vid felaktig sökning.

## Slutsats

De automatiserade UI-testerna som implementerats för Bothniabladets bildhanteringssystem har spelat en central roll i valideringen av två grundläggande användningsområden: inloggning och bildsökning. Med hjälp av moderna testverktyg har det varit möjligt att skapa robusta, repeterbara och modulära testfall som går att köra oberoende av varandra. Detta har i sin tur bidragit till en ökad förståelse för systemets beteende under olika förutsättningar, inklusive felhantering, tillgänglighet och stabilitet.

Att använda tre olika verktyg – Playwright, Cypress och Selenium – har även gett insikter i hur olika tekniker kan komplettera varandra beroende på användningsområde, utvecklingsmiljö och teststrategi. Där Playwright och Cypress möjliggör kodnära och programmeringsorienterade tester, möjliggör Selenium i Java att definiera automatiserade UI-tester med hjälp av WebDriver och strukturerade testmetoder.

Även om vissa brister identifierades, såsom “case-sensitive”-sökning och tillgänglighetsproblem i felmeddelanden, visar projektet tydligt hur automatiserade tester kan bidra till förbättrad kvalitet, ökad kontroll över systemutveckling och bättre användarupplevelse. Slutsatsen är att automatiserad testning inte bara är en rekommendation utan en nödvändighet i moderna utvecklingsprojekt.