NTNU
Det skapende universitet

Verktøy for evaluering av brukergrensesnitt for bibliografisk informasjon

Erlend Klakegg Bergheim

Master i informatikk
Oppgaven levert: Juli 2010
Hovedveileder: Trond Aalberg, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Verktøy for evaluering av brukergrensesnitt for bibliografisk informasjon

Erlend Klakegg Bergheim

12. juli 2010

Innhold

1 Oppsummering

I denne oppgaven skal jeg se på logging generelt og logging på Verdensveven spesielt for å lage et verktøy som kan logge brukeraktiviteter i en helt vanlig nettleser uten å benytte ekstra utstyr.

I første omgang gjennomføres en mindre teoristudie om logging, og logging på Verdensveven spesielt, før de to verktøyene lages. Det må også velges teknologi som passer til formålet og som lar applikasjonene kunne settes opp i etterkant av ferdigstillelse. Hovedvekten av oppgaven blir å se på hvilke tekniske muligheter man har ved logging i en nettleser samt implementasjon av de to verktøyene.

Som resultat er det laget et verktøy som logger brukeraktiviteter i moderne nettlesere uten å måtte benytte andre hjelpemidler. Det er også laget en mindre applikasjon som passer til målgruppen slik at verktøyet kan testes ut i praksis.

2 Innledning

I denne oppgaven skal jeg se på logging generelt og logging på Verdensveven spesielt for å lage et verktøy som kan logge brukeraktiviteter i en helt vanlig nettleser uten å benytte ekstra utstyr. Ved å lage et verktøy som lett kan brukes til å brukerteste et nettsted kan gjøre det lettere å samle inn data, og det er derfor viktig at verktøyet som lages oppleves enkelt samtidig som det logger nok til at dataene kan benyttes i etterkant.

For å komme i havn skal det i tillegg til verktøy for logging utvikles en frittstående applikasjon som fungerer som brukergrensesnitt mot bibliografiske data for å kunne teste at loggeverkøyet fungerer som det skal.

I første omgang gjennomføres en mindre teoristudie om logging, og logging på Verdensveven spesielt, før de to verktøyene lages. Det må også velges teknologi som passer til formålet og som lar applikasjonene kunne settes opp i etterkant av ferdigstillelse. Hovedvekten av oppgaven blir å se på hvilke tekniske muligheter man har ved logging i en nettleser samt implementasjon av de to verktøyene.

2.1 Oppgavetekst

I et forskningsprosjekt hvor vi samarbeider med noen bibliotek om utvikling av nye datamodeller for bibliografisk informasjon har vi behov for et verktøy for å kunne analysere og evaluere forskjellige dialoger, presentasjon og interaksjonsformer

I denne oppgaven skal studenten utvikle et brukergrensesnitt mot en XML-database som inneholder bibliografisk informasjon (Java eller C++). Målet er et verktøy som støtter forskjellige typer søke, navigering og presentasjonsformer (bla. forskjellige typer hierarkiske trefflister)

I oppgaven skal studenten identifisere og prøve ut forskjellige teknikker/metoder for å logge brukeroppførsel, f.eks. måle tiden det tar å fullføre en oppgave eller tiden som brukes på hvert trinn i informasjonssøkeprosessen etc.

3 Teori

3.1 Logging

Det finnes flere måter å gå frem på for å finne ut hvordan et produkt brukes. Kunnskap om hvordan et produkt benyttes kan gjøre en i stand til å endre produktet, forbedre produktet, endre markedsføring, endre salgskanaler eller til og med slutte å tilby produktet.

Jeg skal her se på en del tilnærminger til hvordan man kan finne ut hvordan et produkt benyttes, først med fokus på nettsider og deretter mer generelt. I tillegg er det interessant å se hva man kan gjøre før produktet brukes for å forbedre det[2].

3.1.1 Logger

Tidlig begynte man å benytte visualisering og statistikk basert på tjenerlogger til å finne brukermønster for nettsider. Dette var noe som blomstret spesielt i perioden 1995-2000, og spesielt innen nettsider som er laget rundt søk kom man tidlig til at det finnes tre ulike typer brukere:

- Brukere - besøkende som aktivt gjennomfører søk
- Gjester - besøkende som ikke gjennomfører søk
- Speidere - søkemotorer som er innom for å indeksere nettstedet

Spesielt introduksjonen av speidere ser i litteraturen ut til å ha tatt enkelte prosjekter på senga, til tross for at speidere allerede på denne tiden normalt identifiserte opprinnelsen slik at de kunne filtreres vekk fra loggene om ønskelig.

En av de store ulempene med tjenerlogger er at mengden informasjon som kan innhentes er begrenset til tidspunkt, hvilken nettside, hvilken nettleser som var benyttet, adressen forespørselen kom fra og eventuelt hvilket nettsted brukeren nettopp besøkte. Loggen sier ingen ting om ting som:

- Hvilken link brukeren klikket på (lokasjon på siden)
- Hvor lang tid brukeren brukte på en aktivitet
- Om brukeren var tilstede hele tiden
- Om det er samme bruker gjennom hele sesjonen

Flere av ulempene nevnt over eksisterer fordi protokollen som World Wide Web basserer seg på, Hypertext Transfer Protocol (HTTP), ikke støtter ”state”, noe som gjør at man i prinsippet ikke kan gjenkjenne brukere. Dette unngår man på sider med innlogging ved hjelp av kjeks (cookies) og gjerne en teknikk som kalles for ”sesjoner” (sessions).

I tillegg tilsa utvikling av World Wide Web og brukermassen at det i samme periode også ble introdusert proxyer og deling av IP i større skala enn tidligere. Proxyer gjør at sider kan mellomlagres på en tjener mellom netttjeneren og klienten slik at brukeren får bedre opplevelse av nettstedet, men dette fører også til at oppslag ikke logges.

Deling av IP gjorde til at flere brukere kunne aksessere samme nettsted samtidig, og at alle aksessene så ut til å komme fra samme bruker siden samme adresse kom i loggen for alle (man kan se for seg at nettleseregenskaper kan hjelpe til med å skille mellom sesjoner, men i en homogent miljø lar dette seg ikke gjøre).

3.1.2 Ekstern logging

En del av ulempene rundt bruk av logger er i dag løftet vekk fra tjenerloggen og over til nettstedet og eksterne tjenesert. En av de mer utbredte tjenestene er i dag Google Analytics, et verktøy som er basert på JavaScript og som blir tatt i bruk ved å legge inn en snutt fra Google på alle sidene man ønsker å spore.

Den store ulempen med denne typen tjenester er at nettstedet må endres for å kunne ta tjenesten i bruk, men dette overskygges i mange tilfeller fordelene med de ekstra egenskapene som kan hentes inn:

- Brukerens støtte for teknologi som Flash og Java
- Skjermstørrelse, skjermfarger og språk
- Får vite hvor lenge en gitt sidesesjon var aktiv
- Får vite hvor stor andel som er tilbakevendende brukere

I tillegg klarer tjenesten å skille de ulike brukersesjonene fra hverandre siden snutten fungerer på klient-siden i stedet for tjener-siden, og man kan da benytte seg av kjeks og sesjoner som i utgangspunktet ikke er en naturlig del av HTTP.

Utover Google Analytics finner man flere leverandører, samt muligheten til å sette opp egen ekstern logging ved hjelp av åpen kildekode-prosjektet Piwik.

I dette og forrige avsnitt er egenskaper som brukerlokasjon (land), tilkoblingshastighet og annet som kan innhentes basert på IP ikke tatt med da dette i utgangspunktet er felles for begge at dette lar seg gjøre.

3.1.3 Intervju

En måte å finne ut hvordan et nettsted oppleves er gjennom intervju. Dette er en godt kjent måte å innsamle informasjon om brukere for å evaluere et produkt, og brukes mye utenfor IT-bransjen.

Dette er en ikke-teknisk metode som består typisk i at en intervjuer stiller åpne spørsmål til en bruker, og svarene brukes som grunnlag for vurdering av et produkt. Produktet kan være både et fysisk produkt, for eksempel en stol eller en ost, men også imatrielle produkter som programvare og nettsteder (på mange måter to sider av samme sak).

Under intervju kan man velge om man ønsker å skrive ned svarene man får, eller om man ønsker å ta opp samtalen på bånd/video. Det er opp til intervjuobjeektet å godta eventuelle opptak.

Når denne metoden benyttes er det typisk gått noe tid fra brukeren benyttet et produkt til man skal fortelle hva man tenker om det, noe som gjør at brukeren kan ha glemt

ting, for eksempel ting som man irriterte seg over og hva man egentlig gjorde, men man kan også endre mening i løpet av perioden, man irriterte seg kanskje grønn over at det var laget et altfor simpelt verktøy, men når man fikk tid til å tenke på det var det kanskje godt allikevell. At man forandrer mening underveis i en slik prosess er vanskelig å fange opp undrer et intervju.

3.1.4 Overråkning

I likhet med intervjuer er ikke dette en metode som er forbeholdt IT-bransjen. Dette er også en veldig vanlig måte å finne ut noe om hvordan dyr reagerer. I fortsettelsen tar jeg utgangspunkt i at det er mennesker som overvåkes.

Konseptet er veldig enkelt; man setter opp et kamera og filmer det som skjer/ikke skjer. Man har i grunn to muligheter, å fortelle brukeren at man tar opp eller å la være (skjult kamera).

I forbindelse med overvåkning må man også finne ut hvordan man stiller seg etisk. Å ikke si ifra til brukeren at man overvåker kan i verste fall få juridiske konsekvenser.

Overvåkning er i utgangspunktet en god måte å samle inn data på, men selv om man ser en ting er det ikke sikkert at det er nettopp det man ser (høh?). En handling kan være forårsaket av andre grunner enn overvåker forestiller seg, og kan gi feil utslag.

3.1.5 Spesialisert overvåkning

I stedet for å overvåke "alt" kan man velge å fokusere kun på gitte ting. Innen IT er man gjerne interessert i å kunne finne ut hva en person ser på, da kan man finne ut hva som fanger oppmerksomheten og hva som går en "huset forbi".

"Eyetracking" er en kjent metode for å overvåke/spore øyets bevegelse over en skjerm. Gjennom dette kan man se hvilke deler av skjermen brukeren ser, og man kan finne ut hva som får oppmerksomhet og hva som ikke får oppmerksomhet. Et eksempel på forskning som er gjort ved hjelp av en "eye-tracker" er å finne ut at man vanligvis leser nettsider i en "F", altså at man gjerne ikke ser det som er nederst til høyre på skjermbildet maan presenteres. (Jakob Nielsen)

3.1.6 Syntaks og verifisering

I stedet for å følge brukeren kan man ta grep for å sikre en best mulig opplevelse. Det finnes flere måter å gjøre dette på:

- Validere (X)HTML og CSS
- Eliminere døde lenker
- Brukbarhet

Noe av det enkleste å teste er at et nettsted følger standarder, for eksempel (X)HTML og CSS, ved å validere nettstedet på ressurser som W3C sin (X)HTML-validator og CSS-validator. For stadig flere standarder finner man validatorer, av de som kan nevnes er Microformats, Atom og RSS.

Besøkende opplever døde lenker som negativt, og det er utviklet programvare for å verifisere at lenker, både utgående og interne, fungerer og ikke kommer opp med feil. Det ikke alle tenker på å teste er at referanse til innhold (bilder, lyd, utseende og ressurser) også kan forsvinne eller bli utilgjengelig i løpet av et nettsteds levetid.

Brukbarhet har de siste 10 årene kommet stadig mer i fokus, og når for eksempel Norge nå innfører lover som tilsier at blant annet offentlige nettsteder må være tilpasset også brukergrupper med spesielle behov (blinde/svaksynte, ulike former for fargeblinde og funksjonshemmede) kan man ta i bruk allerede eksisterende verktøyer for å simulere bruk for de ulike brukergruppene. Utover dette eksisterer det også retningslinjer for hvordan man skal evaluere brukerhet på offentlige nettsteder.

3.1.7 Assistert utvikling

Gjennom å bygge inn "tilbakemelding" i verktøy som brukes under utvikling kan man veilede utviklere underveis i utviklingen og dermed kunne få et bedre produkt i utgangspunktet før eventuell brukertesting.

Slike verktøy har sett dagens lys, men følger man litteraturen så dukket dette opp på 1990-tallet, men ser ut til å være forvitret underveis. Det kan tenkes at semantikken for vanlige nettsteder tok igjen disse produktene.

4 Design

Jeg skal lage et verktøy som kan samle inn data for analyse og evaluering basert på en brukers oppførsel i et lukket system, eksempelvis et grensesnitt som gir tilgang til bibliografisk informasjon. Under uvikling av et slikt verktøy er det noen spørsmål man må spørre seg:

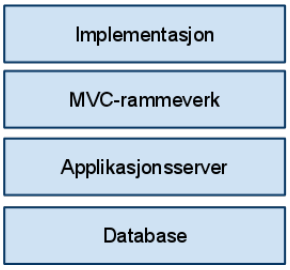
- Skal produktet gjenbrukes?
- Hvor høy skal terskelen være for å gjenbruke?

I dette tilfellet finner jeg det naturlig at dette verktøyet skal kunne gjenbrukes i den formen det leveres eller kunne videreutvikles med tanke på bredere gjenbruk. En god måte å underbygge muligheten for gjenbruk er å senke terskelen for å være i stand til å gjenbruke verktøyet.

De siste årene er det blitt betydelig lettere å utvikle nettsteder, og en stor fordel med disse er at det ikke er knyttet noen spesiell teknologi til oppbyggingen bortsett fra at produktet kommuniserer over HTTP og sender ut et innhold som kan forståes av nettlesere. Under dette kan man benytte det meste, og etter at rammeverket Ruby on Rails kom på banen er det dukket opp veldig mange rammeverk i de alle utbredte programmeringsspråk som gjør det mulig å raskt utvikle relativt avanserte og spesialiserte løsninger.

Verktøyet jeg skal lage skal kunne leve i nettleseren og la brukeren gjennomføre testene på i alle moderne nettlesere, noe som vil gjøre det mulig å gjenbruke verktøyet uavhengig av den underliggende programvaren som skal testes.

4.1 Arkitektur



Figur 1: Applikasjon med flere lag.

Oppbyggingen av applikasjonen er i utgangspunktet tenkt veldig tradisjonelt gjennom å benytte en lagdelt arkitektur. Dette er også tvingende nødvendig siden det skal kjøres kode både på serversiden og på klientsiden.

4.1.1 Kode i nettleseren

For å spore aktivitet er man nødt til å ty til skript som kjører på klientsiden, i dette tilfellet JavaScript. JavaScript har sin opprinnelse fra nettleseren Netscape, og etter

at det var uglesett, inkludert av forfatter, har dette skriptingspråket fått en ny vår takket være "AJAX" og oppblomstringen av rike applikasjoner som kjører i en helt vanlig nettleser.

JavaScript er ikke den eneste muligheten for å kjøre kode i nettleseren. Microsoft var tidlig ute med sitt alternativ til JavaScript, Visual Basic Script (VBS), og er støttet av Microsoft sin nettleser, epost-lesere og operativsystem. Den meste kjente koden skrevet i dette skriptspråket er sannsynligvis ILoveYou-viruset. VBS er ikke støttet i andre nettlesere enn Internet Explorer.

Mens VBS kjører som en del av nettsiden (kan jobbe med elementene på nettsiden) er dette ikke et alternativ for alternativene Flash, Silverlight og Java Applet. De forskjellige teknologiene har varierende utstrekning, hvor Flash regnes for å være den med best støttede. Felles for disse er at de opererer programmer kjørende på en nettside, og har begrensede muligheter til å endre nettsiden den kjører på. Tidligere var det enkelte som lurte på om man skulle benytte Java Applet til å gjennomføre AJAX, siden Java Applet tillater å kalle og å kalles fra JavaScript, men dette har man gått bort fra.

Selv om dette ikke inngår som egen del i tegningen over er dette en del av implementasjonen siden siden dette ikke er et frittstående produkt.

4.1.2 MVC-rammeverk

De siste årene har man sett en oppblomstring av rammeverk som skal lette utvikling av nettapplikasjoner, og det er derfor naturlig å velge et rammeverk som passer til oppgaven. Målet med rammeverkene er oftest å gjemme vekk en del detaljer, samt utnytte potensiale som ligger i ulike programmeringsspråk. En liten, og på ingen måte komplett, liste over noen alternativer:

- Ruby on Rails - Rammeverket som var med på å sparke i gang oppblomstringen. Man skriver applikasjonen i Ruby, men takket være JRuby og IronRuby kan man kompilere applikasjonene til å kjøre på både Javas JVM og Microsofts CLI.
- Django, Pylons - Django har en egen tilnærming til fordeling av ansvar (ikke MVC), men hører allikevel til i gruppen over rammeverk for Python. Også disse rammeverkene kan kjøre på JVM og CLI takket være henholdsvis Jython og IronPython, men det er varierende støtte for dette.
- Zend Framework - ZF er utviklet av selskapet bak PHP, Zend, og kom på mange måter som et svar på populariteten til Ruby on Rails og dens bruk av MVC.
- ASP.Net MVC - Microsoft sin tilnærming, og er begrenset til å kjøre på programvare levert av selskapet.
- JavaServer Faces - JSF er i utgangspunktet et bibliotek for generering av nettsider (HTML), men takket være mulighetene som ligger i JSF kombinert med Java Enterprise Edition 6 er dette et rent Java-alternativ.
- Grails - Rammeverk som benytter programmeringsspråket Groovy, et språk som kompileres til å kjøre på JVM. Det er veldig mange likheter mellom Ruby on Rails og Grails.

Siden det skal benyttes Java er det nærliggende å i det minste velge et rammeverk som kjører som Java (på JVM), og da står man igjen med rammeverk alle rammeverkene opplistet unntatt ASP.Net MVC (PHP kan serves på JVM takket være bibliotek som Quercus). Siden målet ikke er å komme seg vekk fra Java i seg selv, så er Grails er nærliggende alternativ uten å måtte ta steget helt inn i JEE-verden. Groovy sin syntaks er prikk lik Java, men ulikt Java er det i virkeligheten et dynamisk skriptspråk som også har en del alternativ syntaks.

Kort fortalt prøver Groovy å kutte ned i nødvendig kode samtidig som den tilgjengeliggjør utvidet funksjonalitet som lages av Java-utviklere hver eneste dag, samtidig som det gir støtte for alt Java støtter. Det er også viktig å titte på brukermiljøet, og både Groovy og Grails har etablerte brukermiljø og utgitte bøker, og populariteten er stadig voksende.

4.1.3 Applikasjonstjener

Apache sin HTTP-server er i dag den mest utbredte applikasjonsserveren for Verdensveven. En klassiker er LAMP-platformen som består av Linux, Apache, MySQL og PHP/Python/Perl, hvor Apache opererer som en applikasjonsserver for de ulike programmeringsspråkene. Primært er Apache en HTTP-tjener som server statiske nettsider, men takket være utvidelser kan man benytte den som applikasjonsserver, proxy og annet.

En måte man gjerne velger å drifte sine Java-løsninger på er ved å velge en applikasjonsserver for Java, for eksempel Glassfish og JBoss, som driftes bak en Apache satt opp som proxy og med moduler for mellomlagring slått på for å minske lasten. Man kan også velge å sette applikasjonsserveren direkte på nett uten Apache foran.

4.2 JavaScript

Til tross for den høye alderen til JavaScript finnes det per i dag ikke noen komplett og korrekt implementasjon av språket, og støtten er varierende. Det var i mange år preget av en proprietær implementasjon gjort av Microsoft for deres nettleser. De siste årene har flere rammeverk som JQuery og Prototype gjort det mulig for utviklere å slippe å ta hensyn til hvilken nettleser man kjører i takket være en stor innsats som er lagt ned i rammeverkene.

Rammeverkene har ikke bare gjort det mulig for utviklere å lage grunnleggende funksjonalitet i rike applikasjoner som kjører i nettleseren, det blir stadig vanligere å komme med både effekter og utseende slik Script.acol.us (tillegg til Prototype), JQuery UI (tillegg til JQuery), Google Web Toolkit (GWT) og ExtJS.

4.2.1 AJAX

”Asynchronous JavaScript and XML”, eller AJAX, er en teknikk som bredte seg som ild i tørt gress rundt 2005, og er en teknikk som har gjort oppblomstringen av rike applikasjoner i nettlesere mulig.

Det er viktig å merke seg at AJAX egentlig er et navn på en gruppe med teknikker, og er således ikke knyttet direkte til teknologi selv om navnet tilsier dette. Den består av tre deler:

- Klienten, en nettside, foretar en asynkron forespørsel til serveren.
- Klienten mottar data av noe slag.
- Klienten gjør seg nytte av mottatte data.

På det første punktet benyttes normalt et objekt i JavaScript som heter XMLHttpRequest som åpner for å gjøre kall fra en nettside. Selve objektet trenger hverken å gjøre jobben asynkront eller ved hjelp av XML. Her bygger klienten typisk en adresse og eventuelle ekstra data som sendes med for å hente data.

Det andre punktet går ut på at nettsiden får svar fra klienten, og typisk at en funksjon trigges med de mottatte dataene. I AJAX brukte man i begynnelsen XML, men siden XML er unødvendig verbose og JavaScript sin støtte for lesing av XML var varierende dukket alternativet JavaScript Object Notation (JSON) opp.

JavaScript har en veldig konsis måte å uttrykke objekter og matriser, noe JSON utnytter i tillegg til funksjonen eval(). I dag er det uvanlig å ikke finne et JSON-alternativ i data fra API og det er laget biblioteker for alle mulige programmeringsspråk for å kunne jobbe med JSON. Det er til og med en del av standardpakkene i Google sitt Android-operativsystem.

I tillegg til å være data som er bygget opp på en gitt måte kan data som sendes tilbake til klienten være helt vanlig HTML, noe som gjerne benyttes av vev-rammeverk hvor man har eget AJAX-bibliotek. Eksempel på dette er siste versjon av JSF som følger med Java Enterprise Edition 6.

På det siste punktet er det opp til klienten å gjøre det man ønsker med data. Om det er helt vanlig HTML vil man vanligvis ønske å oppdatere et element med denne informasjonen, eller hvis det ikke er HTML vil man benytte den til å generere HTML. Det er ikke alltid man ønsker å benytte data som kommer i retur til noe.

4.2.2 Cross Site Scripting (XSS)

Når man benytter AJAX er det ikke alltid man ønsker å hente data kun fra egen server, men gjerne også andre servere. Det er her lagt inn en begrensninger i dagens nettlesere som gjør at AJAX mot andre adresser (sudomene) enn den siden kommer fra ikke lar seg gjøre.

Noe av grunnen til disse begrensingene er at man fant ut i 2003/2004 at man ved hjelp av XSS kunne sende informasjon tilbake til en server slik at dette kunne medføre misbruk og kriminelle handlinger. Det man gjorde var å sende innholdet i sesjonsnøkkelen tilbake, noe som gjorde det mulig for angripere å komme inn på sesjonen som brukeren, og få tilgang til det samme som brukeren. Basert på rettigheter man har oppnådd (administrator av en nettside, nettbank, osv) kunne man gjennomføre angrep.

For å omgå begrensningene laget man en mønster man kaller for JSONP, "JSON with padding", som gjør at man dynamisk setter inn et ekstra script-element på nettsiden og sender med som parameter en metode som skal kalles. Her benytter man ikke

AJAX, men nettleseren sin måte å introdusere JavaScript gjennom eksterne filer. Når skriptet åpnes innholder det JSON (som jo er JavaScript av natur) som mates inn i metoden som er definert.

Man kan spørre seg om grunnen til at det er lagt inn begrensninger på XSS i nettleseren når det allikevell er mulig å omgå disse, noe som er begrunnelsen for at man også har sett på muligheten for å la XMLHttpRequest får gjøre kall utenfor egen adresse ved å legge på ekstra informasjon i header, men dette er foreløpig kun vært testet i en tidligere beta-utgave av Firefox og trukket tilbake i etterkant.

4.3 Henting av nettsider

Hypertext Transfer Protocol (HTTP) er i utgangpunktet en stateless protokol, men til tross for dette var man tidlig ute med muligheter for å la den støtte state. Dette gjøres ved at det hektes på kjeks (cookies) når det innhold sendes i retur til nettleseren som etterspør innhold. Kjeks brukes til å lagre innhold som kan gjenbrukes over flere besøk, men samtidig er det svært utbredt å benytte dette til å ta vare på brukersesjoner eller andre sesjoner for den gitte brukeren mellom flere forespørsler.

Ved brukertesting av et nettsted kan kjeks gjøre at informasjon henger igjen mellom to tester. Dersom man tenker seg testing av en handlevogn hvor man gjerne ønsker tømme denne mellom ulike test-scenarioer så har man to ulike måter å gjøre dette på. Dersom nettsider hentes av JavaScript som kjører loggingen og denne nettsiden henter fra samme server, så vil man være nødt til å identifisere og fjerne eventuelle kjeks som ikke tilhører logge-programvaren. Den andre muligheten er å hente nettsider ved hjelp av JavaScript gjennom en proxy som kjører på serveren, hvor kjeks lagres i test-sesjonen og kan tømmes mellom hver enkelt tekst.

En eventuell introduksjon av en proxy gir også en annen fordel, man kan nå hente nettsider fra andre adresser og la test-sesjonen ta vare på kjeksene. Hvis man introduserer tre muligheter for henting av nettsider for testing; ingen proxy, enkel proxy, delvis proxy og full proxy.

4.3.1 Ingen proxy

Alternativet ”ingen proxy” innebærer at det ikke er tilgang på en proxy for nettsider, noe som gjør at eventuelle kjeks er vanskeligere å fjerne siden disse vil blande seg blant de test-programvaren benytter. Man kan da velge å la logge-programvaren holde orden på hvilke kjeks den benytter for å fjerne resten, eller man kan spesifisere hvilke kjeks man forventer at en side skal opprette og heller fjerne målbevisst.

Dette alternativet gjør at man er låst til samme server (host) som logge-programvaren, noe som gjør at eventuel oppdeling av logge-programvare og programvare som skal testes må være eller se ut til å være på samme server.

4.3.2 Enkel proxy

En enkel proxy håndterer ikke kjeks, men henter alle sider som om det var første besøk hver gang. Med en slik proxy vil det i utgangspunktet ikke være mulig å ”huske brukeren” mellom sidene, en løsning som egner seg godt når det skal testes nettstedet som inneholder kun ”statisk” innhold.

Selv om innhold som går gjennom proxy ikke kan knyttes kjeks til vil innhold som ikke går gjennom proxy kunne ha kjeks knyttet til seg på lik linje med vanlig bruk.

4.3.3 Delvis proxy

Med mellomalternativet gjør man det samme med kjeks fra egen server som man ville gjort under ”ingen proxy”, men i tillegg åpner man for å kunne hente nettsider fra andre servere, noe som i utgangspunktet gjør at man kan fri seg fra å ha programvaren som skal testes på samme server som test-programvaren.

Ved hjelp av en proxy kan man la brukersesjonen i test-programvaren ta vare på kjeksene fra ulike servere slik at brukeren vil oppleve at også eksterne nettsteder ”nullstilles” mellom tester. I utgangspunktet kan man også velge å la proxy operere uten state, noe som gjør at brukeren ikke kan logge seg inn eller lagre ting mellom nettsidene på det eksterne nettstedet.

4.3.4 Full proxy

Det som skiller ”delvis proxy” og ”full proxy” er at med sistnevnte vil også nettsider på samme server som test-programvaren hentes gjennom en proxy slik at håndtering av kjeks gjennomføres i test-sesjonen, og man kan da sikre at de korrekte kjeksene fjernes (og i utgangspunktet også kunne legges til) mellom test-scenarioer.

4.3.5 XSS og proxy

Som man ser kan man ved hjelp av en proxy omgå begrensningene som er laget på grunn av redselen for Cross Site Scripting (XSS). Det er verdt å merke seg at dette gjelder utelukkende nettsiden som ”fyller nettleseren”, men om denne nettsiden benytter rammer (frames) eller interne rammer (iframe) så vil innholdet i disse ikke nødvendigvis gå gjennom proxy uten å la proxy endre adresser til disse sidene i innholdet som sendes til nettleseren.

4.3.6 Annet innhold

Ofte tenker man at det er nettsidene som er de eneste som kan endre seg mellom sesjoner, men dette er feil. Dersom man tenker seg de gamle telleverkene som viser antall besøkende på en nettside, så benyttet de gode telleverkene en liten kjeks til å huske deg slik at man ikke ble telt flere ganger på samme side. Siden det er nettsidene som er vanskelig å hente fra andre servere (på grunn av XSS) vil man i utgangspunktet ikke

hente alle elementene på nettsiden ved hjelp av proxy, noe som gjør at informasjon som er lagret ved hjelp av, i dette tilfellet, et telleverk (bruker bilde-elementer på nettsiden) ikke telle oppover etterhvert som man bytter test-scenario siden denne kjeksen blir lagret i nettleseren og krever aktiv handling fra brukeren for å kunne fjerne.

4.3.7 Misbruk av proxy

Dersom man ønsker å ta i bruk en proxy er det viktig at denne ikke kan misbrukes. En proxy som ikke er begrenset på noen måte kan potensielt misbrukes til å kamuflere angrep på andre nettsider. For å hindre misbruk kan man legge begrensninger som at brukeren må holde på med et test-scenario for at den skal kunne returnere nettsider, samt at man kan begrense hvilke servere hvert enkelt test-scenario skal kunne hente nettsider fra.

4.4 Sider med JavaScript

Rike applikasjoner på verdensveven kan i denne oppgaven deles inn i to grupper, de hvor aktivitet kan spores og resten. Applikasjoner som i utgangspunktet ikke kan spores er applikasjoner hvor funksjonaliteten er laget ved hjelp av Flash, Silverlight eller Java. Det er viktig å merke seg at man her fokuserer på den delen som er synlig for besøkende. Selv om det er noen som ikke lar seg spore, så kan også applikasjoner som får funksjonalitet fra JavaScript by på problemer, og jeg ønsker her å se på tre ulike problemer og en tilnærming til løsning.

4.4.1 Benytter samme variabelnavn

Det er to elementer i JavaScript som byr på problemer. For det første finnes det ikke navnerom (namespace) i språket, noe som medfører at om samme variabelnavn opprettes på to ulike steder på en side, så vil den ene overskrive den andre, uavhengig av innhold og uten advarsler. Den andre er at alt er variabler, noe som betyr at variabler også benyttes til å navngi funksjoner.

Begge disse sidene ved JavaScript kan gi problemer på en vanlig nettside, men kan gi enda større problemer i et loggeverktøy som skal bruke to sett med kode, sin egen kode og den andre siden sin kode. I en løsning hvor man bruker to sett med kode øker sannsynligheten for at man overskriver hverandres kode, og i dette tilfellet vil det alltid være loggeverktøyet sin kode som overskrives siden denne lastes først.

Rammeverk for JavaScript har vært opptatt av denne problemstillingen lenge, og har i utgangspunktet falt på den samme løsningen (ihvertfall de store). Løsningen er at man oppretter et objekt som man legger funksjonaliteten i. Dette objektet er ikke et helt vanlig objekt som man kjenner fra programmeringsspråk, men er snarere et bidrag til JavaScript sin tilnærming til objektorientert programmering, slik at dette objektet kan brukes til å instansiere nye objekter av samme type ved behov.

4.4.2 Overskrive event-håndtering

Ikke alle applikasjoner som bruker hendelsehåndteringen til JavaScript bruker den nye modellen, noe som gjør at de baserer seg på den gamle hvor det kun er mulig med én hendelse per element. Når loggeverktøyet laster en nettside for visning kan hendelsehåndtering overskrives, enten av testverkøyet eller av verktøyet som lastes inn.

Ved en overskrivelse av hendelsehåndtering er to mulige scenarioer:

- Loggeverktøyet overskriver funksjonalitet på nettsiden som skal testes og brukeren kan oppleve at nettsiden ikke responderer som forventet.
- Nettsiden som skal testes overskriver funksjonalitet på loggeverktøyet, noe som medfører at mekanismer laget for å logge brukeroppførsel slutter å fungere og gjør at testobjektet opplever at loggeverktøyet feiler og sender brukeren ukontrollert ut av testingen.

4.4.3 Spore bakgrunnsoppgaver

I et loggeverktøy vil det være naturlig at logg sendes til applikasjonen ved hjelp av AJAX, men på en nettside som skal logges er det i utgangspunktet mulig at også denne nettsiden ønsker å gjennomføre mot egen applikasjon. Det er her to vanskeligheter:

- Dersom det forsøkes å gjøre AJAX mot en annen serveren en den loggeverktøyet kjører på vil dette feile, noe som brukeren kan oppleve som at applikasjonen som skal testes feiler.
- Det er vanskelig å logge hva som skjedde med siden etterhvert som brukeren fikk opp informasjon i bakgrunnen.

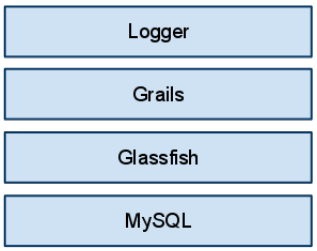
4.4.4 Slå av JavaScript

Ved siden av å programmere JavaScript til loggeverktøyet på en slik måte at det ikke ”konkurrerer” med testsider er det å skru av JavaScript på nettsider som skal testes en annen tilnærming som kan gi resultater i mange tilfeller. Man kan vedhjelp av regulære uttrykk fjerne innhold på nettsider som består av JavaScript og dermed fjerne muligheten for at det kan oppstå problemer mellom loggeverktøy og applikasjonen som skal testes.

Å slå av JavaScript på applikasjoner er ikke en mulighet dersom applikasjonen legger til grunn at det skal være slått på, men da har man fortsatt potensielle utfordringer som er nevnt i de tre foregående punktene.

5 Implementasjon

5.1 Arkitektur



Figur 2: Aritektur for logge-verktøyet

I dette prosjektet ønsker jeg å ta utgangspunkt i Glassfish som applikasjonsserver for en Grails-applikasjon. Glassfish er utviklet av Sun, og siste utgave støtter fullt ut Java Enterprise Edition 6. I teorien skal man fra JEE6 ikke trenge å endre på applikasjoner for å bytte applikasjonsserver takket være standardisering av resursdeling (JNDI).

Som bonus kan man ved hjelp av få klikk installere Grails-bibliotekene på Glassfish slik at man ikke trenger å legge disse med applikasjoner som skal serves på applikasjonsserveren.

For database faller valget mitt på MySQL av flere grunner:

- MySQL er lett tilgjengelig for de som måtte ønske å benytte programvaren.
- MySQL er et kraftig relasjonsdatabase system og grunnleggende funksjonalitet her legger tilrette for de behov som er for lagring av logger.
- For å sette opp en databaseserver som gjør det mulig å benytte applikasjonen som skal lages trenger man minimale kunnskaper, og for mange Linux-distribusjoner er dette gjerne kun én kommando unna.
- Jeg er godt kjent med MySQL etter mange års bruk.

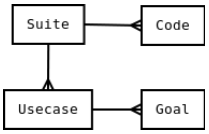
Selv om jeg velger MySQL som databasesystem kan dette byttes ut uten videre takket være konfigurering i applikasjonsserveren, og alle databasesystemer som går overens med Java Persistence API kan benyttes.

5.2 Administrasjon av tester

For å administrere testene er en del av implementasjonen bestående av et grafisk administrasjonsgrensesnitt som benyttes rett i nettleseren. Når applikasjonen er installert finner man dette grensesnittet i undermappen ”admin” i forhold til applikasjonens rot.

Testene er langt på vei bygget opp etter inspirasjon fra xUnit[1], hvor man setter opp et testsett (suite) som inneholder flere tester som kan gjennomføres i en vilkårlig rekkeføl-

ge. I denne implementasjonen benyttes testsett (suite) som den naturlige inndelingen, hvor brukerhistorier (use case) ansees å være en enkelt test.



Figur 3: Oppbygging av testsett

For å detektere at testeren har klart en oppgave benyttes nettadresser som mål (goal). Når brukeren kommer til en adresse som er definert som mål for en gitt brukerhistorie sendes man videre til neste. For at brukeren skal kunne kjøre testene er det lagt opp til at man benytter en adresse hvor en nøkkel (code) sendes inn som parameter.

5.3 Funksjonalitet

5.3.1 Kontroll på brukeren

For å kunne logge brukerens oppførsel er det i utgangspunktet nødvendig å ta en viss kontroll over funksjonaliteten på nettsidene som skal testes. Dette gjøres ved å føre brukeren til en side hvor innhold brukeren skal benytte lastes inn samtidig som adressen til nettsiden endrer seg etter #-tegnet i adressen. På den måten opplever brukeren at det skjer noe i adresselinjen samtidig som det skjer noe i nettleseren.

Innhold som skal presenteres oppdateres med lyttere ved hjelp av JavaScript hver gang det lastes slik at brukeren ikke forlater siden. For å hente innhold benytter jeg en innebygget proxy slik at dette kan tilbys som tjeneste gjennom AJAX.

5.3.2 Proxy

I denne oppgaven er det valgt å implementere det som tidligere er kalt en ”enkel proxy”. Det forutsettes at nettsteder som programvaren er beregnet på ikke har behov for å lagre kjeks på brukeren i og med at innhold ofte kan ansees å være forholdsvis statisk i brukergrensesnitt for bibliografisk informasjon.

Siden det forutsettes at man kan kjøre testene på applikasjoner som endrer seg i liten eller ingen grad underveis i livsløpet mens den testes er det ikke laget noen form for lagring av sider, noe som kan medføre at siden brukeren og testleder ser kan variere dersom informasjonen i applikasjonen endrer seg i større eller mindre grad fra testen gjennomføres til testen evalueres.

5.3.3 Logge hendelser

JavaScript har i løpet av de siste årene innført en bedre hendelsesmodell enn tidligere, en modell som gjør det mulig å ha mer enn en funksjon til å lytte på en gitt hendelse fra et gitt element. Dessverre er ikke denne modellen implementert i alle nettlesere,

enda, men rammeverk som JQuery har valgt å implementere modellen for nettlesere som ikke har støtten eller mangler deler av den.

Hendelser logges ved å generere et objekt som ved hjelp av AJAX sendes til serveren for å lagres i databasen. Dette objektet består av tre ulike data:

- Tidspunkt - tidspunkt i millisekund for når hendelsen inntraff.
- Type - hvilken type hendelse logges.
- Data - de fleste hendelsene har data, hvor dette gjøres om til en JSON-streng som kan puttes direkte i databasen.

Ved logging av hendelser benyttes klokken på testerens maskin for å hindre at det blir feil tidspunkt på hendelsene fordi lasten på serveren som tar imot logg-innslagene skulle bli for høy til at den klarer å ta unna i forventet tempo.

Ulike former for spesialisert logging benytter ulike hendelser i nettleseren til å starte, noe applikasjonens logging allerede er et tegn på.

Logging av mus-posisjon er en form for bruk av hendelser i nettleseren. JavaScript har en hendelsemodell hvor alle alle foreldreelementene til et gitt element også blir trigget, og dette kan utnyttes til å legge triggeret "onmouseover" på body-elementet, og hver gang man flytter musen i vinduet vil da dette triggeret trigges. For at denne hendelsen skal trigges er brukerens mus nødt til å være inne i nettleseren i den delen hvor nettsider presenteres. Med en gang brukeren beveger seg utenfor dette området eller bytter fane vil det ikke logges noen hendelser.

Skjema logges ved å hekter lyttere på de ulike elementene som utgjør et skjema, som tekstfelt, passordfelt, knapper og lister, og kan deretter logge aktiviterene og innholdet i dem etterhvert som de endrer status. I tillegg må elementene på forhånd indekseres for å kunne skille dem fra hverandre i loggen.

I tillegg lar det seg gjøre å hente hva som er synlig i nettleseren til enhver tid gjennom å finne ut hvor stort vinduet er samt hvor mye forflytning det på siden. Det er ingen trigger for dette, men dette kan for eksempel logges sammen med museposisjon.

5.4 Presentasjon av logg

Det er lagt opp til tre ulike visningsformer for loggen. To av formene er gjennom ren uthenting av logg, mens den tredje er et forsøk på å blåse liv i loggen gjennom å visualisere det brukeren gjorde under testingen.

5.4.1 Detaljert visning

Den detaljerte visningen er den mest omfattende utlistingen, og er enkelt og greit råmaterialet som er samlet inn. Under utvikling av applikasjonen har denne visningen vært viktig i forbindelse med å kontrollere at innkomne data gir riktig resultater.

5.4.2 Enkel visning

Den enkle visningen skiller seg fra den detaljerte ved at grupper av like typer innslag i loggen oppføres kun med det første innslaget. Eksempelvis vil hundre musposisjoner da kunne reduseres til kun ett innslag, noe som gjør loggen lettere å lese og også lettere å følge gangen i hva brukeren har gjort.

5.4.3 Replay

I et forsøk på å visualisere det brukeren kunne se på sin skjerm kan man følge med på testen uten å sitte ved siden av brukeren. Gjennom å visualisere loggen gjennom hendelser på skjermen kan man følge med etterhvert som brukeren flytter seg rundt på nettstedet og gjennomfører oppgaven som er gitt.

Det er potensielt mange feilkilder til denne, her er noen av de viktigste:

- Brukeren har justert nivået av zoom i nettleseren, noe som kan medføre at det ikke detekteres riktig egenskaper i loggen.
- Nettsiden ble ikke lastet riktig på grunn av mangel på elementer eller annet innhold som er midlertidig borte eller fjernet.
- Nettstedets utseende er ikke optimalisert for alle nettlesere, noe som gjør at

Ingen av feilkildene som er nevnt er det prøvd å veie opp for siden dette er tidvis umulig/vanskelig å detektere, eller for krevende å skulle prøve å detektere til at det gir mening å løse problemet i denne oppgaven.

5.5 Videreutvikling

5.5.1 Innlogging for administrasjon

En naturlig utvidelse vil være å ilegge administratorer som skal organisere tester mulighet til å begrense tilgangen til administrasjonspanelet. Dette er funksjonalitet som er utelatt fordi man kan se for seg ulike tilnærminger til tilgangskontroll og innsyn, i tillegg er dette funksjonalitet som er lett å lage i Grails.

5.5.2 Flere former for mål

Ved gjennomføring av brukerhistoriene som utgjør testsettene kan det tenkes at man ønsker å trigge at en brukerhistorie er fullført på andre måter enn gjennom å komme til en adresse. Eksempel på dette kan være ved testing av en nettbutikk hvor man ønsker å se om brukeren klarer å få en gitt kombinasjon av varer i handlekurven eller tilsvarende. Dette er noe som potensielt vil kunne løses ved at siden som testes kan kjøre trigger eller at man kjører et skript som kontrollerer en status på nettsiden.

I tillegg til trigging på gitte situasjoner ved hjelp av skript i nettleser kan man se for seg at man ønsker å benytte regulære uttrykk til å definere adresser som gyldige målsider. Eksempel på nytte kan være at man skal finne en bok skrevet av Henrik Ibsen, og

dersom adressen inneholder frasen '/HenrikIbsen/' kan man tenke seg at si at man har kommet i mål med oppgaven.

5.5.3 Valg av proxy

I Design er det skissert en rekke alternativer for hvordan systemet kan støtte proxy, men kun en av dem er implementert. I et tilfelle hvor det implementeres en eller flere vil det være naturlig å la det være opp til administrator av testene får muligheten til å velge hvilken politikk man ønsker åp benytte, noe som i prinsippet kan variere fra brukerhistorie til brukerhistorie.

5.5.4 Endre innholdet

En av fordelene med proxy er at man har mulighet til å legge inn kode som eksempelvis fjerner all JavaScript på en nettside før den sendes videre til testeren. Man kan se for seg at dette ville være en fordel dersom det skal testes et nettsted hvor logge-applikasjonen ikke klarer å logge nok til at man kan forstå hva brukeren har gjort i etterkant.

5.5.5 Kontrollere til replay

Implementasjonen av replay-visning av logg baserer seg på en implementasjon som aktivt hindrer at nettleseren fryser under presentasjon. Alternative implementasjoner av replay i JavaScript vil benytte samme eller tilsvarende funksjonalitet for å visualisere, men gjennom å benytte en Just In Time-innmating av materialet som skal presenteres kan man se for seg at det skal kunne tas pause og spole i loggen ettersom man selv ønsker dette.

I tillegg til kontrollere kan man se for seg å benytte eksempelvis sidelastinger som meny til visningen slik at man kan raskt hoppe i visningen.

6 Testing og evaluering

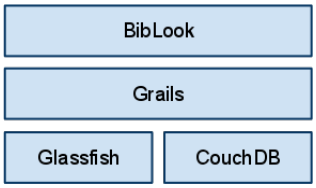
For testing og evaluering av logge-programvaren skal det lages et grensesnitt mot en database med bibliografisk informasjon. Jeg ønsker å ta utgangspunktet som er gjort av Trond Aalberg et al. med utvikling av verktøy for generering av FRBR-poster fra MARC-poster.

6.1 Bibliografisk verktøy

Applikasjonen som skal lages skal være en eksempelimplementasjon av programvare logge-programvaren er beregnet på å bruke, og skal ha ulike former for presentasjon av data og ulike måter å navigere i dataene.

6.1.1 Arkitektur

Arkitekturen for denne programvaren minner om programvaren for logging da det er naturlig at begge applikasjonene kan installeres på samme applikasjonsserver.



Figur 4: Aritektur for test-applikasjon

I stedet for å støtte seg til en XML-database er det her benyttet CouchDB, en dokumentdatabase som benytter JSON som struktur på dataene. CouchDB er en av databasesystemene som er med i hypen "NoSQL" som har blitt stadig mer synlig de siste par årene. Dokumentdatabaser er på ingen måte noe nytt, og både XML- og JSON-databaser er eksempler på denne typen databaser.

Grunnlaget for å velge CouchDB til denne applikasjonen er flere:

- Man gjør spørringer mot CouchDB over HTTP, og behovet for programvarebiblioteker og spesielt oppsett minsker.
- Resultatene fra CouchDB kommer som JSON, og kan lett gjenbrukes direkte i grensesnittet om det er ønskelig å gjøre artige kunster med JavaScript.
- Man kan i etterkant gjennomføre indeksering av data ved hjelp av Lucene for ekte søkefunksjonalitet gjennom å sette opp et innstikk.
- CouchDB er veldig enkelt å installere og å sette opp på Ubuntu.
- For å gjøre spørringer mot databasen (annet enn å hente enkeltdokumenter) må dette genereres på forhånd, og denne indeksen (B+-tre) holdes oppdatert etterhvert som dokumenter oppdateres.

6.1.2 Implementasjon

Datagrunnlaget som ble benyttet er en mindre del av en database med bibliografisk materiale fra Slovenia som på forhånd er gjort om til XML. For å gjøre dette tilgjengelig i CouchDB ble det laget et lite skript som konverterte dataene uten å miste noe av informasjonen under konvertering.

Det fokuseres på to ulike måter å navigere i applikasjonen. Den første er den desidert mest utbredte på Verdensveven; gjennom å klikke på lenker. Den andre måten å navigere på er gjennom å skrive i et felt hvor navn i databasen er lovlige verdier, og man får opp en alfabetisk liste over forslag etterhvert som man skriver (auto complete).

Den alfabetiske listen over forslag er en indeks i CouchDB som er forhåndsgenerert, noe som gir raskt oppslag. Grunnet tidvis varierende kvalitet på data i de ulike feltene vil det på enkelte oppslag dukke opp mulige treff som kunne vært rensket, men siden det ikke er noe spesielt mønster som går igjen er det ikke gjort noe for å luke ut disse.

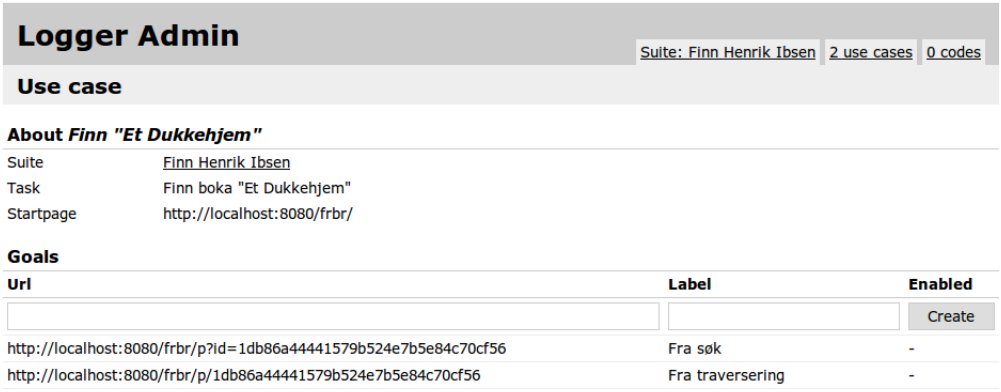
Applikasjonen bygges opp ved hjelp av to ulike sider, en forside og en side for visning. Siden for visning viser de ulike oppslagene med de nødvendige strukturene. Det er her ikke noen direkte presentasjon av enkeltelementene i databasen, men en innhenting av elementer som er nødvendig for å gi brukeren nødvendig informasjon om elementene man besøker.

6.2 Logging

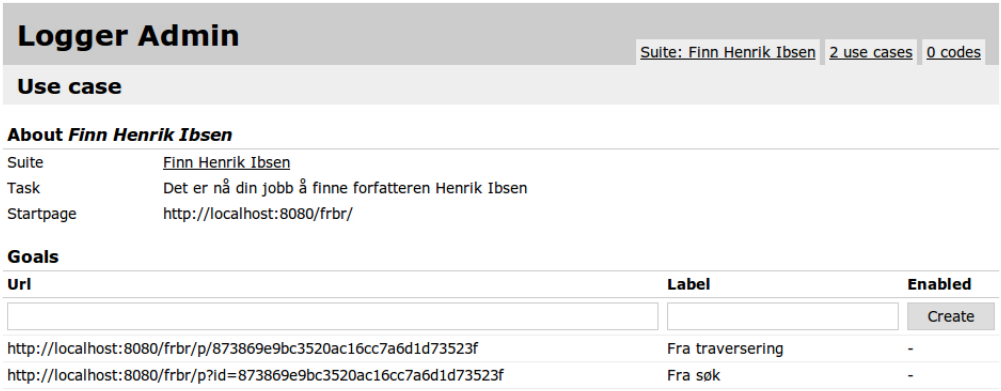
For å teste verktøyet er det laget to brukerhistorier i et testsett.

Figur 5: Vi har opprettet et testsett.

I tillegg har vi to brukerhistorier.



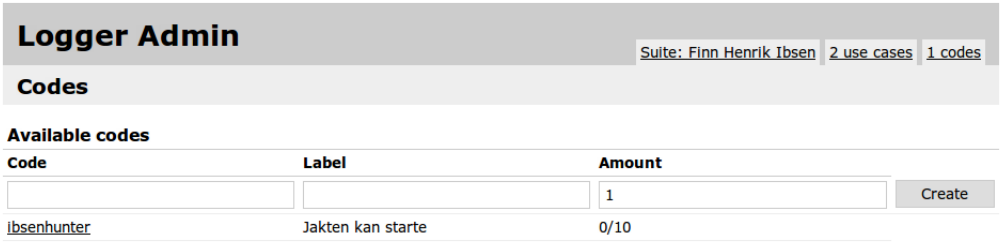
Figur 6: Brukerhistorie 1



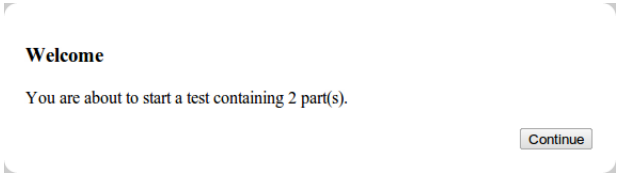
Figur 7: Brukerhistorie 2

For å kunne gjennomføre testene må det også genereres en nøkkel som skal brukes til å starte testene.

Når vi klikker på lenken får vi først en liten velkomst.



Figur 8: Nøkkel er generert.



Figur 9: Velkommen

På neste bilde får vi vite oppgaven.

Figur 10: Oppgave 1

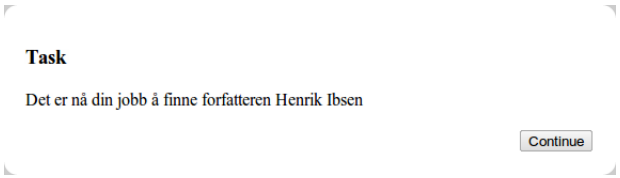
Neste er at vi kommer inn i den lukkede visningen med forsiden til BibLook oppe.

Figur 11: Forsiden

Vi søker på på "Ibsen" og trykker oss videre til Ibsen.

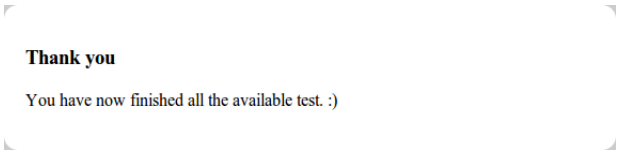
Figur 12: Henrik Ibsen

Med en gang vi trykker på tittelen "Et dukkehjem" blir vi sendt videre til neste oppgave.



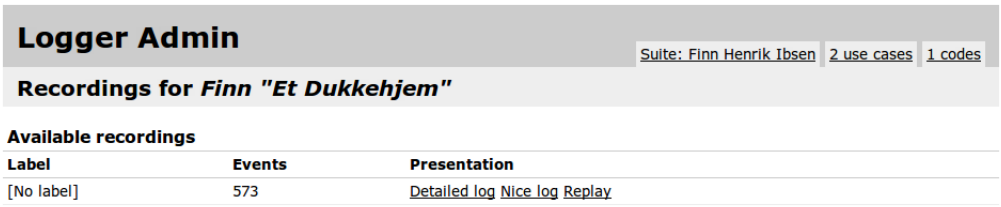
Figur 13: Oppgave 2

Her søker vi igjen på Ibsen, og nå vi har klikket på navnet kommer vi videre til avslutningen av loggingen.



Figur 14: Vi er ferdig.

Når vi kommer inn i admin-grensesnittet kan vi se at det er gjort et opptak.



Figur 15: Et opptak av oppgaven.

Vi kan titte på den detaljerte loggen.

Logger Admin

Suite: Finn Henrik Ibsen | 2 use cases | 1 codes

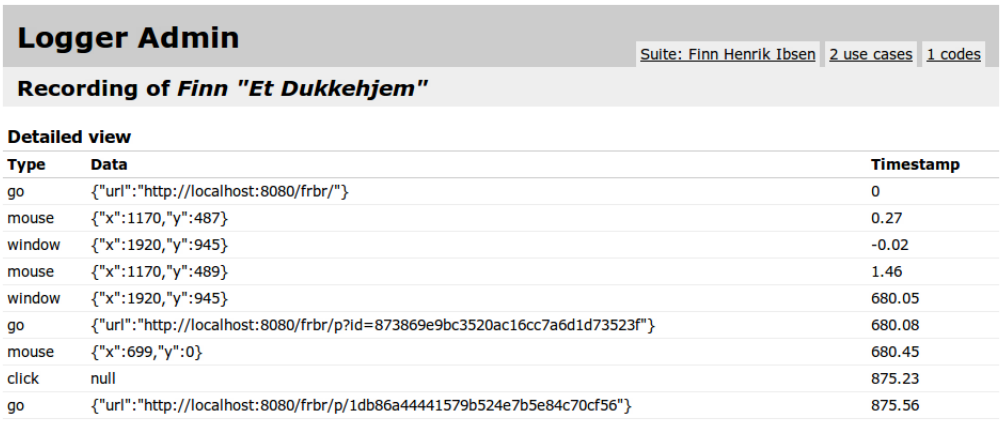
Recording of *Finn "Et Dukkehjem"*

Detailed view

Type	Data	Timestamp
go	{"url":"http://localhost:8080/frbr/"}	0
mouse	{"x":1170,"y":487}	0.27
mouse	{"x":1140,"y":404}	0.13
window	{"x":1920,"y":945}	-0.02
mouse	{"x":1170,"y":489}	1.46
mouse	{"x":1170,"y":488}	1.21
mouse	{"x":1170,"y":489}	163.7
mouse	{"x":1170,"y":489}	176
mouse	{"x":1170,"y":489}	242.94
mouse	{"x":1170,"y":489}	242.85
mouse	{"x":1172,"y":490}	246.9
mouse	{"x":1174,"y":490}	246.98
mouse	{"x":1176,"y":491}	247.06
mouse	{"x":1176,"y":473}	249.22
mouse	{"x":1176,"y":489}	249.06
mouse	{"x":1176,"y":490}	248.9
mouse	{"x":1176,"y":486}	249.09
mouse	{"x":1176,"y":482}	249.14
mouse	{"x":1176,"y":463}	249.3
mouse	{"x":1174,"y":451}	249.38
mouse	{"x":1172,"y":435}	249.6
mouse	{"x":1167,"y":393}	249.65
mouse	{"x":1165,"y":364}	249.7
mouse	{"x":1157,"y":276}	249.95
mouse	{"x":1155,"y":249}	250.11
mouse	{"x":1162,"y":335}	249.79
mouse	{"x":1160,"y":305}	249.86
mouse	{"x":1164,"y":7}	251.48
mouse	{"x":1166,"y":1}	251.54
mouse	{"x":1150,"y":60}	254.75
mouse	{"x":1156,"y":25}	254.65

Figur 16: Detaljert logg

Vi kan titte på den enkle loggen.



Logger Admin

Suite: Finn Henrik Ibsen | 2 use cases | 1 codes

Recording of *Finn "Et Dukkehjem"*

Detailed view

Type	Data	Timestamp
go	{"url":"http://localhost:8080/frbr/"}	0
mouse	{"x":1170,"y":487}	0.27
window	{"x":1920,"y":945}	-0.02
mouse	{"x":1170,"y":489}	1.46
window	{"x":1920,"y":945}	680.05
go	{"url":"http://localhost:8080/frbr/p?id=873869e9bc3520ac16cc7a6d1d73523f"}	680.08
mouse	{"x":699,"y":0}	680.45
click	null	875.23
go	{"url":"http://localhost:8080/frbr/p/1db86a44441579b524e7b5e84c70cf56"}	875.56

Figur 17: Enkel logg

Eller vi kan få loggen vist for oss.

BibLook Find

Person (4.6)

Henrik, Ibsen

Has created

- Gengangere
- Et dukkehjem

Created by Erlend Klakegg Bergheim.

Figur 18: Replay in action.

Legg merke til firkanten som markerer musens posisjon over lenken som ble klikket på.

7 Konklusjon

I denne oppgaven har jeg sett på logging generelt og logging på Verdensveven spesielt. Størst vekt er det lagt på å lage et verktøy for å logge brukeroppførsel utelukkende gjennom å benytte funksjonalitet som er tilgjengelig i en moderne nettleser. For å komme i mål har jeg sett på en del tekniske muligheter som er tilgjengelig i nettlesere, hvor fokus har vært utelukkende på JavaScript siden dette er den laveste terskelen for brukere i dag.

Som resultat er det laget et verktøy som logger brukeraktiviteter i moderne nettlesere uten å måtte benytte andre hjelpemidler. Det er også laget en mindre applikasjon som passer til målgruppen slik at verktøyet kan testes ut i praksis.

Referanser

[1] Wikipedia: xunit. http://en.wikipedia.org/wiki/XUnit.

[2] Melody Y. Ivory and Marti A Hearst. The state of the art in automating usability evaluation of user interfaces. *ACM Comput. Surv.*, 33(4):470–516, 2001.